# 第一章 前端

## 第一节：HTML

### 一、基本介绍

1. HDML—超文本标记语言（HyperText Markup Language） 是一个国际标准（W3C-国际万维网组织(www.w3.org)） `eg:TCP/IP`，用于构建网页内容。
2. 三种技术：HTML、CSS、JS 围绕前端， **浏览器** `Chrome\FireFox\Opera\Safari\IE-Edge` 网页就会产生浏览器兼容性的问题，不同的浏览器厂商不一定100%覆盖兼容W3C标准，因此我们需要做浏览器兼容性测试， 在软件公司里，可以限定用户使用那种浏览器。
3. HTML由谁来执行？——浏览器 Browser

- B/S结构：Browser/Server(浏览器/服务器)【主流】Java做的就是B/S结构 语言：Java、PHP、Python、ASP.NET
- C/S结构：Client/Server (客户端/服务器) 代表：迅雷、酷狗、QQ 语言：C++、C#(WinForm)、
- 单机软件：eg:Office 语言：VB、C++(VC++)、C#、Java、

### 二、HTML常用属性

- HTML5前一个版本是HTML4.01不是HTML4
- `<meta charset="UTF-8">` 声明网页的元数据
- 标签：是网页浏览器识别符
- 元素：从开始标签开始到结束标签结束包含中间内容称为元素

#### （一）基本元素

1. `<html>` HTML的根元素，H5允许完全省略这个元素
2. `<head>` H5文档的也买你头部分，也可以完全省略
3. `<title>` 定义文档的页面标题
4. `<body>` 定义文档的页面主体部分
5. `<h1>` 到 `<h6>` 定义标题1到标题6
6. 注意：几乎所有的HTML元素都可以指定：id、class、style等核心属性
7. `<br/>` 插入一个换行
8. `<hr/>` 定义一个水平线，也叫分割线
9. `<!---->` 定义注释
10. `<p>` 定义段落
11. `<div>` 定义文档中的节
12. `<span>` 与 `<div>` 基本相似，区别是该标签定义的节默认不会换行。本身不会改变任何东西，只为CSS样式服务

- 单标签：自闭合标签，只为实现自身的功能，而不封装数据，自己在内部开始在内部结束，注意书写格式---斜杠写在后面

#### （二）文本格式化元素

> 按照自定义的格式进行重现

1. `<b>` 定义粗体文本
2. `<strong>` 定义粗体文本，表示很重要的文本，与 `<b>` 标签类似,效果相同，强调强度不同

3. `<i>` 定义斜体文本
4. `<em>` 定义强调文本，实际效果和斜体差不多
5. `<u>` 定义下划线
6. `<small>` 定义小号字体文本，常用于小号印刷体
7. `<sup>` 定义上标文本
8. `<sub>` 定义下标文本
9. `<bdo>` 定义文本显示方向，dir属性值为ltr或者rtl用来控制文字

```
<!--rtl: right to left-->
<bdo dir="rtl">这是一首简单的小情歌</bdo>
```

## （三）语义相关元素

> 需要知道的是`<q>、<del>、<ins>、<pre>`四个，其它了解即可

1. `<address>` 用于表示一个地址
2. `<blockquote>` 用于定义一段长的引用文本
3. `<q>` 用于定义一段短的引用实体，浏览器会自动添加双引号
4. `<cite>` 用于表示作品的标题
5. `<code>` 用于表示一段计算机的代码
6. `<dfn>` 定义一个专业术语
7. `<del>` 定义文本中被删除的文本,通常结合插入标签一起使用
8. `<ins>` 定义文本中插入的文本
9. `<pre>` 预格式化文本,完全安装自己写的格式进行显示
10. `<kdb>` 用于定义就按盘文本，该元素用于表示文本是通过键盘输入
11. `<var>` 用于表示一个变量

```
<body>
<!--<q>:用于指定短的应用文本，浏览器自动添加双引号-->
    有句话这么说来着：<q>人这一辈子，有两句肉麻的话是非说不可的。 --'谢谢你'和'对不起'。</q><br/>
<!--<del>、<ins>-->
    《遇见》被<del>孙燕姿</del><ins>张敬轩</ins>翻唱过……<br/>
<!--pre：预格式化文本-->
    <pre>
    小情歌
    这是一首简单的小情歌
    唱着人们心肠的曲折
    我想我很快乐
    当有你的温热
    脚边的空气转了
    </pre>
</body>
```

## （四）图像相关元素

1. src：指定图片文件的所在位置，可以是相对路径也可以是绝对路径
2. alt：该属性指定一段文本，作为图片的提示信息
3. height：指定图片的高

4. width：指定图片的宽

```
<img src="../img/朱茵.jpg" alt="朱茵-紫霞仙子朱茵3.jpg" height="540px" width="960px"/>
```

5. usemap：属性用于链接的map的映射
6. `<map>`：用于定义图片映射，包含一个或多个子元素

- eg：中国地图，点击某个省就对应切换成某个省的地图

7. `<area/>`：用于定义图片映射的内部区域，不用于封装数据

- shape：指定内部区域是那种区域
- coords：指定多个坐标值，用于确定区域位置

```
<!--加载图片,使用map映射-->

<img src="../img/朱茵.jpg" alt="紫霞仙子.jpg" width="960px" height="540px" usemap="#map"/>
<!--创建一个map标签-->

<map id="map">
    <area shape="circle" coords="480px, 270px, 300px" href="introduce.html"/>
</map>
```

## （五）超链接和锚点

1. 标签

- href：指定超链接所关联的两一个资源

```
//href 和 src 的区别
src是直接引入进行显示
href则是需要跳转才能显示
```

- target：指定使用框架中的哪一个框架来装载另一个资源（本窗口还是新建一个窗口进行显示）
- a元素可以包含文本、图像、各种文本格式化元素和表单元素等内容

```
<a href="demo1.html">链接到demo1</a>
<a href="https://www.baidu.com/">百度一下，你就知道！</a>
<a href="../img/朱茵.jpg">朱茵照片</a>

<!--target属性，新窗口打开-->
<a href="https://www.baidu.com/" target="_blank">百度一下，你就知道！</a>
```

2. 锚点的演示

```
<body>
    <a id="top">顶部位置</a>

    <img src="../img/朱茵.jpg" alt="紫霞仙子.jpg"/><br/><br/>
    <img src="../img/朱茵.jpg" alt="紫霞仙子.jpg"/><br/><br/>
    <img src="../img/朱茵.jpg" alt="紫霞仙子.jpg"/><br/><br/>

    <a href="#top">回到顶部</a>
</body>
```

## （六）列表相关元素

1. `<ol>`定义有序列表，也只能包含`<li>`子元素

- start：指定列表的起始数字，默认是第一个
- type：指定使用哪种类型编号

2. `<ul>`定义无序列表，只能定义`<li>`子元素
3. `<li>`定义列表项
4. `<dl>`也用于定义列表，包含`<dt>`和`<dd>`两种子元素
5. `<dt>`定义标题列表项
6. `<dd>`定义普通列表项
7. 列表的嵌套

```
<!--有序列表-->
<ol start="3" type="i">
    <li>侯马</li>
    <li>孝义</li>
    <li>运城</li>
    <li>晋中</li>
</ol>

<!--无序列表-->
<ul type="circle"><!--circle:空心圆  square:实心方形-->
    <li>篮球</li>
    <li>足球</li>
    <li>乒乓球</li>
    <li>羽毛球</li>
</ul>
```

## （七）表格相关元素

1. `<table>` 用于定义表格
2. `<caption>`用于定义表个标题，0或1个
3. `<thead>`定义表格头，0或1个

- `<th>`定义表格的页眉单元格

4. `<tbody>`定义表格体，可以多个

- `<tr>`用来定义行

5. `<tr>`定义行，`<td>`定义列

- colspan:跨列

- rowspan:跨行
- hight:指定单元格的高度
- width：指定单元格的宽度

6. `<table>` 中可以设置的属性

- cellpadding:指定单元格内容和单元格边框之间的间距
- cellspacing:指定单元格之间的距离
- width:指定表格的宽度

```
<table border="1px" width="400" align="center" cellpadding="20px" cellspacing="0px">
    <caption>值日表</caption>
    <thead>
        <th>周一</th>
        <th>周二</th>
        <th>周三</th>
    </thead>
    <tbody style="text-align: center">
        <tr>
            <td colspan="2">张三</td>
            <td rowspan="2">王五</td>
        </tr>
        <tr>
            <td>赵六</td>
            <td>孙七</td>
        </tr>
    </tbody>
</table>
```

## （八）框架相关元素

- 因为 `<frameset>` 和 `<body>` 不兼容，所以第一步删掉 `<body></body>`
- 搭建框架 `<frameset>`

1. `<fameset>` ---不推荐使用，已被淘汰

- frame
- rows:分割行,cols:分割列

```
<frameset rows="20%, 80%">
    <frame src="top.html"/>
    <frameset cols="30%, 70%">
        <frame src="left.html">
        <frame src="right.html" name="right">
    </frameset>
</frameset>
```

2. `<iframe>` ---经常使用

```
<body>
    <iframe width="30%" height="700px" src="left.html"></iframe>
    <iframe width="69%" height="700px" src="right.html" name="right"></iframe>
</body>
```

### （九）H5新增

1. 使用canvas元素
2. 使用audio和vdeo元素,表示视频资源

- src:配置视频的路径
- controls:显示播放控件
- autoplay:表示当前页面加载完成之后自动加载
- loop:表示视频循环播放
- width:视频的宽度
- height:视频的高度

> 当属性值唯一且与属性名一致，只需要写一个属性名即可

```
<body>
    <video src="../video/RM动画宣传.mp4" width="640px" height="360" controls autoplay loop>
</video>
</body
```

## 二、表单控件

### （一）表单元素

- `<from>`元素用于生成输入表单

### （二）使用input元素

- 单行文本框：text
- 密码输入框：password
- 单选框：radio
- 复选框：checkbox
- 文件上传域：file
- 提交按钮：submit
- 重置按钮：reset
- 单纯的重置按钮：button

```
<form>
    <!--表单控件-->
    用户名：<input type="text" name="username" value=""/><br/>
    密码：<input type="password" name="password" value=""/><br/>
    性别：<input type="radio" name="sex" value="boy"/>男
         <input type="radio" name="sex" value="girl"/>女<br/>
    爱好：<input type="checkbox" name="hobby" value="qin"/>琴
         <input type="checkbox" name="hobby" value="qi"/>棋
         <input type="checkbox" name="hobby" value="shu"/>书
         <input type="checkbox" name="hobby" value="hua"/>画<br/>
    上传文件：<input type="file" name="file" value=""/><br/>
    <input type="submit" value="提交按钮"><br/>
    <input type="reset" value="重置按钮"><br/>
    <input type="button" value="单纯按钮"><br/>
</form>
```

### （三）input中包含的属性

- checked：默认被选中
- disabled：禁用元素/标签
- maxlength：制定文本框中所允许输入的最大字符数
- placeholder：占位符
- readonly：设置只读，不允许修改，只针对文本框
- size：设置元素宽度

```
<form>
    <!--表单控件-->
    提示：<input value="个人注册信息" readonly size="50" style="text-align: center"/><br/>
    用户名：<input type="text" name="username" value="" placeholder="用户名/邮箱/手机号" /><br/>
    密码：<input type="password" name="password" value="" minlength="6" maxlength="8"
placeholder="用户密码"/><br/>
    性别：<input type="radio" name="sex" value="boy" checked/>男
         <input type="radio" name="sex" value="girl"/>女<br/>
    爱好：<input type="checkbox" name="hobby" value="qin"/>琴
         <input type="checkbox" name="hobby" value="qi"/>棋
         <input type="checkbox" name="hobby" value="shu" checked/>书
         <input type="checkbox" name="hobby" value="hua"/>画<br/>
    上传文件：<input type="file" name="file" value=""/><br/>
    <input type="submit" value="提交按钮"><br/>
    <input type="reset" value="重置按钮"><br/>
    <input type="button" value="单纯按钮"><br/>
</form>
```

### (空) 强调：使用注意

1. `<input>`使用除按钮外，一般都需要添加name、value属性
2. form表单提交有两种提交方式：get和post
    - get：提交时在地址栏显示提交信息（默认使用get方式提交）
    - post：提交时不会在地址栏显示提交信息（一般选用此提交方式）
3. form表单提交后需要进行跳转---action属性

```
<form method="post" action="address.html">
<!-- …… -->
</form>
```

### (四) 使用button定义按钮

- 优点：可以嵌套图片

1. disable：指定是否禁用此按钮
2. name：指定按钮的唯一名称
3. type：指定按钮属于那种按钮

```
<button type="submit">提交按钮</button><br/>
<button type="reset">重置按钮</button><br/>
<button type="button">单纯按钮</button><br/>
```

### (五) 列表和下拉列表

1. `<select>`元素

- size：指定该列表中框中可同时显示多少个列表项

2. `<option>`用于定义列表框选项或菜单项

- selected：默认选中

3. `<optgroup>`用于定义列表项或者菜单项组

- lable：指定该项组的标签，该属性必填
- disable：置于不可选状态

```
籍贯：<select name="birthplace" size="3">

        <optgroup label="山西" disabled>
            <option value="taiyuan">太原</option>
            <option value="linfen">临汾</option>
            <option value="yuncheng" selected>运城</option>
        </optgroup>

        <optgroup label="山东">
            <option value="qingdao">青岛</option>
            <option value="jinan">济南</option>
            <option value="yantai">烟台</option>
        </optgroup>

    </select><br/>
```

### （六）使用textaea定义文本域

1. cols指定文本域的宽度
2. rows指定文本域的高度
3. rows指定文本域的高度
4. disabled禁用
5. 禁止拉伸：style="resize:none"

```
备注：<br/>
<textarea rows="10" cols="50"  style="resize:none">hello world!</textarea><br/>
```

# 第二节：CSS

## 一、CSS样式的使用

### （一）内嵌样式

- 使用style在开始标签内部书写
- style="属性名：属性值；属性名：属性值；....."

```
<div style="width:100px; height: 100px; background-color: blue">简单爱</div>
<span style="color:red;">小情歌</span>
```

### （二）内部样式

- 在head头部写上一对`<style>`标签

```
<head>
    <meta charset="UTF-8">
    <title>内部样式</title>
    <style type="text/css">
        div{
            width:200px;
            height:200px;
            background-color: red;
        }
    </style>
</head>
<body>
    <div>大本钟</div>
</body>
```

### （三）外部样式

- 在外部新建一个样式（CSS）文件
- 在需要调用CSS文件的HTML页面中的head头部使用`<link/>`标签链接样式文件
- 前端

```
<head>
    <meta charset="UTF-8">
    <title>外部样式</title>
    <link href="3.外部样式.css" type="text/css" rel="stylesheet"/>
</head>
<body>
    <div>123</div>
</body>
```

- CSS

```
div{
    width:300px;
    height:100px;
    background-color: royalblue;
}
```

### （四）样式应用优先级问题

- 采用就近原则
- 内嵌样式优先级最高
- 内部样式和外部样式优先级取决于里元素的距离远近（就近原则）

## 二、CSS选择器

### （一）元素选择器

- 通过标签名来找到想要操纵的元素，进而进行样式的修饰

```
<head>
    <meta charset="UTF-8">
    <title>元素选择器</title>
    <style>
        div{
            color: red;
        }
        span{
            color: green;
        }
        h3{
            color: gold;
        }
    </style>
</head>
<body>
    <div>这是一首简单的小情歌</div>
    <span>唱着人们心肠的曲折</span>
    <h3>我想我很快乐 </h3>
</body>
```

## （二）属性选择器

- E[attr]对具有sttr属性的E元素起作用
- E[attr="value"]对所有包含attr属性并且该属性值为value的E元素起作用
- E[attr^="value"]对具有attr属性，并且属性值是以指定值开头的E元素起作用
- E[attr$="value"]对具有attr属性，并且属性值是以指定值结尾的E元素起作用
- E[attr*="value"]对具有attr属性，并且属性值包含指定值的E元素起作用

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        /*选择包含 id 属性的所有元素*/
        a[id]{
            background-color: red;
        }
        /*选择包含 class 属性的所有元素*/
        a[class]{
            background-color: green;
        }
        /*选择 class 属性值为 "c" 的所有元素*/
        a[class="c"]{
            background-color: gold;
        }
        /*选择 href 属性值以 "http" 开头的所有元素*/
        a[href^="http"]{
            color: green;
        }
        /*选择 href 属性值以 "css/" 结尾的所有元素*/
        a[href$="css/"]{
            color: white;
        }
```

```
        /*选择 href 属性值中包含子串 "apple" 的所有元素*/
        a[href*="apple"]{
            color: red;
        }
    </style>
</head>
<body>
    <a href="http://www.w3school.com.cn/" id="1">W3School</a><br/>
    <a href="http://www.w3school.com.cn/css/" class="a">CSS</a><br/>
    <a href="http://www.w3school.com.cn/html/"class="c">HTML</a><br/>

    <hr />

    <a href="http://www.w3c.org/">W3C</a><br/>
    <a href="http://www.microsoft.com" >Microsoft</a><br/>
    <a href="http://www.apple.com.cn">Apple</a><br/>
</body>
```

## (三) id选择器

- 通过#加上id值方式来获取相应的元素

```
<head>
    <meta charset="UTF-8">
    <title>id选择器</title>
    <style>
        #div1{
            color: red;
        }
        #div2{
            color: green;
        }
        #div3{
            color: gold;
        }
    </style>
</head>
<body>
    <div id="div1">这是一首简单的小情歌</div>
    <div id="div2">唱着人们心肠的曲折</div>
    <div id="div3">我想我很快乐</div>
</body>
```

## (四) class选择器

- 通过一个'.'加上类名的方式获取对应的元素

```
<head>
    <meta charset="UTF-8">
    <title>class选择器</title>
    <style>
        div.a{
            color: red;
        }
```

```
        span.a{
            color: green;
        }
        .b{
            color: gold;
        }

    </style>
</head>
<body>
    <div class="a">这是一首简单的小情歌</div>
    <span class="a">唱着人们心肠的曲折</span>
    <h3 class="b">我想我很快乐 </h3>
    <p class="b">当有你的温热</p>
</body>
```

## （五）包含选择器

- 又称后代选择器
- 父元素和子元素之间使用空格分开即可
- 用于指定目标选择器必须处于某个选择器对应的元素内部

```
<head>
    <meta charset="UTF-8">
    <title>包含选择器</title>
    <style>
        p span{
            color: royalblue;
        }
        #p2 .span2{
            color: green;
        }
    </style>
</head>
<body>
    <p>
        这是一首简单的小情歌  <span>唱着人们心肠的曲折</span>
        我想我很快乐  <span>当有你的温热</span>  脚边的空气转了
    </p>

    <p id="p2">
        这是一首简单的小情歌  <span class="span2">唱着我们心头的白鸽</span>
        我想我很适合  <span class="span2">当一个歌颂者</span>  青春在风中飘着
    </p>
</body>
```

## （六）子选择器

- 用于指定不妙选择器必须是某个选择器对应的子元素，孙子元素不可使用
- 而包含选择器只要位于外部选择器对应的元素内部即可使用，即使是孙子元素也可以
- 要求目标选择器必须作为外部选择器对应的元素的直接元素才行

```
<head>
```

```
    <meta charset="UTF-8">
    <title>子选择器</title>
    <style>
        p>span{
            color: royalblue;
        }
        p>span>em{
            color: green;
        }
    </style>
</head>
<body>
    <p>
        这是一首简单的小情歌    <span>唱着人们心肠的<em>曲折</em></span>
        我想我很快乐    <span>当有你的<em>温热</em></span>    脚边的空气转了
    </p>
</body>
```

## （七）CSS3新增的兄弟选择器

- Selector1~Selector2{...}
- 用来匹配Selector1对应的元素的后面的能匹配的Selector2兄弟节点
- 对一类标签产生效果

```
<head>
    <meta charset="UTF-8">
    <title> CSS3新增的兄弟选择器</title>
    <style>
        h1~p{
            color: red;
        }
        #hh~.pp{
            color:blue;
        }
    </style>
</head>
<body>
    <p>这是P标签</p>
    <p>这是P标签</p>
    <p>这是P标签</p>
    <h1 id="hh">这是一级标题</h1>
    <P class="pp">这是一级标题后面的P标签</P>
    <P>这是一级标题后面的P标签</P>
    <P>这是一级标题后面的P标签</P>
</body>
```

## （八）相邻兄弟选择器

- 只对相邻的一个标签产生效果

```
<head>
    <meta charset="UTF-8">
    <title>相邻兄弟选择器</title>
```

```
    <style>
        h1+p{
            color: red;
        }
        h1+p+p{
            color: blue;
        }
    </style>
</head>
<body>
    <p>这是P标签</p>
    <p>这是P标签</p>
    <p>这是P标签</p>
    <h1>这是一级标题</h1>
    <P>这是一级标题后面的P标签</P>
    <P>这是一级标题后面的P标签</P>
    <P>这是一级标题后面的P标签</P>
</body>
```

## （九）选择器组合

- 前端

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link href="13.选择器组合.css" type="text/css" rel="stylesheet"/>
</head>
<body>
    <div id="div_1" class="a">这是一首简单的小情歌</div>
    <span>唱着人们心肠的曲折</span>
    <h3 id="h3_1">我想我很快乐 </h3>
    <pclass="d">当有你的温热</p>
</body>
```

- CSS

```
#div_1,span,#h3_1,.d{
    color: green;
}
```

## （十）伪元素选择器

- first-letter：指定对象内的第一个字符起作用
- first-line：对指定对象内的第一行内容起作用
- before：指定对象内部的前端插入内容
- after：指定对象内部的尾端添加内容

```
<head>
    <meta charset="UTF-8">
    <title>伪元素选择器</title>
    <style>
```

```
        p:first-letter{
            font-size: 20pt;
            color: red;
        }
        p:first-line{
            color: green;
        }
        p:before{
            content: "小情歌--";
        }
        p:after{
            content: "--吴青峰";
        }
    </style>
</head>
<body>
    <P>这是一首简单的小情歌,唱着人们心肠的曲折,
        我想我很快乐,当有你的温热,脚边的空气转了
    </P>
</body>
```

## （十一）伪类选择器

- 子元素选择器
  - first-child
  - last-child
- nth-child选择器
  - nth-child(n)
  - nth-last-child(n)
  - nth-child(odd)设置奇数行
  - nth-child(even)设置偶数行

```
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <style>
        /*设置第一行（父元素的第一个孩子）*/
        li:first-child{
            color: red;
        }
        /*设置最后一行*/
        li:last-child{
            color: blue;
        }
        /*设置指定行（从前数）*/
        li:nth-child(2){
            color: gold;
        }
        /*设置指定行（从后数）*/
        li:nth-last-child(2){
            color: green;
        }
        /*设置奇数行*/
        tr:nth-child(odd){
```

```
            background-color: white;
        }
        /*设置偶数行*/
        tr:nth-child(even){
            background-color: gray;
        }
    </style>
</head>
<body>
    <ol>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
    </ol>
    <ul>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
    </ul>

    <table border="1px" width="300px" cellspacing="0px" style="text-align: center">
        <tr>
            <td>1-1</td>
            <td>1-2</td>
            <td>1-3</td>
        </tr>
        <tr>
            <td>2-1</td>
            <td>2-2</td>
            <td>2-3</td>
        </tr>
        <tr>
            <td>3-1</td>
            <td>3-2</td>
            <td>3-3</td>
        </tr>
        <tr>
            <td>4-1</td>
            <td>4-2</td>
            <td>4-3</td>
        </tr>
        <tr>
            <td>5-1</td>
            <td>5-2</td>
            <td>5-3</td>
        </tr>
    </table>
</body>
```

## （十二）动态伪类选择器

- link：超链接被访问前的状态
- visited：超链接被访问后的状态

  > visited伪类可能会暴露用户浏览的信息记录，攻击者可能会根据此来判断用户曾经访问过的网站，造成不必要的损失，因此浏览器决定限制伪类的功能，所以不是代码的问题二十浏览器方面的限制

- hover：设置鼠标滑过的状态
- active：设置正在被点击时的状态

```
<head>
    <meta charset="UTF-8">
    <title>动态伪类选择器</title>
    <style>
        /*超链接被访问前的状态*/
        a:link{
            /*去掉下划线*/
            text-decoration: none;
            color: black;
        }
        /*超链接被访问后的状态*/
        a:visited{
            text-decoration: none;
            color: gray;
        }
        /*设置鼠标滑过的状态*/
        a:hover{
            text-decoration: underline;
            color: blue;
        }
        /*设置正在被点击时的状态*/
        a:active{
            color: red;
        }
        /*hover应用在<div>上*/
        div{
            width: 100px;
            height:100px;
            background-color:rosybrown;
        }
        div:hover{
            width: 200px;
            height:200px;
            background-color:red;
        }
        /*hover应用在图片上*/
        img{
            width: 200px;
            height: 150px;
        }
        img:hover{
             width: 400px;
             height: 300px;
         }
    </style>
</head>
```

```
<body>
    <a href="http://www.baidu.com" target="_blank">百度一下，你就知道</a><br/>
    <div></div><br/>
    <img src="../../Html/img/朱茵.jpg" alt="朱茵.jpg"><br/>
</body>
```

# 三、通用元素样式

## （一）颜色和单位的使用

1. 颜色

- 以颜色的名字表示颜色
- 以16进制表示颜色
- rgb()表示颜色

```
<head>
    <meta charset="UTF-8">
    <title>颜色和单位的使用</title>
    <style>
        div{
            width: 200px;
            height:200px;
            background-color: red;
            background-color: #008F00;
            background-color: rgb(0,0,255);
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

2. 单位

- px：宽高尺寸
- %：适应网页大小
- pt：字体大小
- em：相对长度单位

## （二）字体相关属性

1. color

```
<!--color-->
<span style="color: red">1.测试文字---color</span><br/>
```

2. font

- font-style
    - normal
    - italic斜体

    - oblique倾斜体
- font-weight
    - ligher更细
    - normal正常
    - bold加粗
    - bolder更粗
    - 还可以使用具体的数值100到900
- font-size
    - 使用
- font-family

```
<!--font-style-->
<span style="font-style: italic">2.测试文字---font-style</span><br/>

<!--font-weight-->
<span style="font-weight: 900">3.测试文字---font-weight</span><br/>

<!--font-size-->
<span style="font-size:20pt">4.测试文字---font-size</span><br/>

<!--font-family-->
<span style="font-family: '华文行楷';">5.测试文字---font-family</span><br/>
```

3. text-decoration(设置下划线)

- none无修饰
- underline下滑线
- line-through中划线
- over-line上划线

```
<!--text-decoration-->
<span style="text-decoration: underline;">3.测试文字---text-decoration</span><br/>

<span style="text-decoration: line-through">3.测试文字---text-decoration</span><br/>

<span style="text-decoration: overline">3.测试文字---text-decoration</span><br/>
```

4. text-transform(设置大小写)

- none不转换
- capitalize首字母大写
- uppercase全部大写
- lowercase全部小写

```
<!--text-transform-->
<span style="text-transform: capitalize;">4.abcdefghijklmnopqrstuvwxyz</span><br/>

<span style="text-transform: uppercase;">4.abcdefghijklmnopqrstuvwxyz</span><br/>

<span style="text-transform: lowercase;">4.ABCDEFGHIJKLMNOPQRSTUVWXYZ</span><br/>
```

5. line-height(设置行高)

```
<!--line-height-->
<p style="line-height: 50px">5.骚灵情歌》是张敬轩的一首粤语歌曲，由周耀辉填词，林健华作曲，张敬轩演唱的流行音乐，被收录于专辑《笑忘书》，发行于2006年10月20日，2007年2月1日，《骚灵情歌》在香港电台中文歌曲龙虎榜周榜单排名第二。</p>
```

6. letter-spacing(设置字间距)

- normal
- 数值+长度单位

```
<!--letter-spacing-->
<span style="letter-spacing: 10px">6.测试文字---letter-spacing</span><br/>
```

7. word-spacing(规定段落中的词间距)

- normal
- 数值+长度单位

```
<!--word-spacing-->
<span style="word-spacing: 100px">7.这是 一段 测试 文字 --- word-spacing</span><br/>
```

8. text-indent(设置段落文本的缩进)

- 数值+长度单位（em）
- 对p标签、div标签起作用，对span标签不起作用

```
<!--text-indent-->
<p style="text-indent: 2em">8.《骚灵情歌》是张敬轩的一首粤语歌曲，由周耀辉填词，林健华作曲。</p><br/>
```

9. text-align(设置居中)

- 对p标签、div标签起作用，对span标签不起作用
- left
- right
- center

```
<!--text-align-->
<p style="text-align: center">9.测试文字---text-align</p><br/>
```

10. text-shadow(设置阴影)

- 一共四个属性值 `style="text-shadow: red 5px 5px 2px"`
- color：指定该阴影的颜色
- xoffset：指定阴影在横向上的偏移
- yoffset：指定阴影在纵向上的偏移
- radius：指定阴影的模糊半径，越大越模糊
- 可以同时添加多个阴影

```
<!--text-shadow-->
<p style="text-shadow: blue 5px 5px 1px">10.《骚灵情歌》是张敬轩的一首粤语歌曲，由周耀辉填词，林健华作曲。
</p>
<p style="text-shadow: red -5px -5px 2px, blue 5px 5px 1px">10.《骚灵情歌》是张敬轩的一首粤语歌曲，由
周耀辉填词，林健华作曲。</p>
```

# 四、块级元素样式

## （一）盒子模型

1. 设置尺寸

- width：设置元素宽度
- height：设置元素高度

2. 设置内边距:内容与边框之间的距离

- padding-top
- padding-right
- padding-bottom
- padding-left
- padding：简写形式

3. 设置外边距:盒子盒子之间的距离

> margin存在塌陷问题，选取两者中大的哪一个

- margin-top
- margin-right
- margin-bottom
- margin-left
- margin：简写形式

> 注：盒子居中`margin: 0 auto;`

- 代码：

```
<head>
    <meta charset="UTF-8">
    <title>盒子模型</title>
    <style>
        div{
            width: 200px;
            height: 200px;
            background-color: blue;
            padding: 20px;
            margin: 50px;
            /*第一种方式*/
            padding-top: 10px;
            padding-right: 20px;
            padding-bottom: 30px;
            padding-left: 40px;

            margin-top: 10px;
            margin-right: 20px;
            margin-bottom: 30px;
```

```
            margin-left: 40px;

            /*第二种方式*/
            padding: 10px 20px 30px 40px;
            margin: 10px 20px 30px 40px;
        }
    </style>
</head>
<body>
    <div></div>
    <div></div>
</body>
```

4. 处理元素内容溢出

- overflow-x
- overflow-y
- overflow：简写形式
    - scroll：滑动条
    - hidden：隐藏

```
<head>
    <meta charset="UTF-8">
    <title>处理元素内容的溢出</title>
    <style>
        div{
            width: 100px;
            height: 100px;
            background-color: bisque;
            margin: 50px;
        }
        #div_1{
            /*使用滑动条方式一*/
            overflow: scroll;
            /*使用滑动条方式二*/
            overflow-x: scroll;
         }
        #div_2{
            /*使用滑动条方式一*/
            overflow: scroll;
            /*使用滑动条方式二*/
            overflow-y: scroll;
        }
        #div_3{
            /*隐藏*/
            overflow: hidden;
        }
    </style>
</head>
<body>
    <div id="div_1">11111111111111111111</div>
    <div id="div_2">1111<br/>1111<br/>1111<br/>1111<br/>1111<br/>1111<br/>1111<br/>
</div>
```

```
    <div id="div_3">123456789123456789123456789</div>
</body>
```

5. 改变元素类型

- 元素类型
  - inline：行内元素 `<span>`

> 不能设置宽高尺寸 不能独占一行

  - inline-block：行内块级元素 `<img>`

> 可以设置宽高尺寸 不能独占一行

  - block：块级元素 `<div>`、`<p>`

> 可以设置宽高尺寸 独占一行

```
<head>
    <meta charset="UTF-8">
    <title>元素类型</title>
    <style>
        span{
            width: 100px;
            height: 100px;
            background-color: gold;
        }
        img{
            width: 100px;
            height: 100px;
            background-color: gold;
        }
        div{
            width: 100px;
            height: 100px;
            background-color: gold;
        }
        p{
            width: 100px;
            height: 100px;
            background-color: gold;
        }
    </style>
</head>
<body>
    <!--inline：行内元素-->
    <span>span标签</span>
    <span>span标签</span>

    <!--inline-block：行内块级元素-->
    <img src="1.png"/>

    <!--block：块级元素-->
    <div>div标签</div>
    <p>p标签</p>
    <p>p标签</p>
```

```
</body>
```

- display：可以转换元素之间的类型

```
    <head>
        <meta charset="UTF-8">
        <title>元素类型</title>
        <style>
            span{
                width: 100px;
                height: 100px;
                background-color: gold;
                display: block;
                margin: 10px;
            }
            div{
                width: 100px;
                height: 100px;
                background-color: gold;
                display: inline-block;
                margin: 10px;
            }
        </style>
    </head>
    <body>
        <span>span标签1</span>
        <span>span标签2</span>
        <span>span标签3</span>

        <div>div标签1</div>
        <div>div标签2</div>
        <div>div标签3</div>
    </body>
```

- 隐藏元素：display：none；
- 设置元素是否可见：visbility
  - visible：可见
  - hidden：隐藏

6. 浮动

- float
  - left
  - right

7. 阻止元素浮动

- clear

## （二）背景相关属性

1. background(设置背景样式)
2. background-color(设置背景颜色)
3. background-image(设置背景图片)

4. background-size(设置图片的大小尺寸)
5. background-position(设置背景图片位置，前提是必须有background-image属性)
6. background-repeat(设置对象的背景图片是否平铺)

- repeat
- no-repeat
- repeat-x
- repeat-y

```
<head>
    <meta charset="UTF-8">
    <title>背景相关属性</title>
    <style>
        div{
            width: 400px;
            height: 400px;
            background-color: rosybrown;
            background-image: url("../../Html/img/朱茵.jpg");
            background-size:300px 200px;
            background-position: 50px 100px;
            background-repeat:no-repeat;
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

## （三）边框相关属性

1. border

- none-无边框
- hidden-隐藏边框
- dotted-点线边框
- dashed-虚线边框
- solid-实线边框
- double-双线边框

2. border-style

- 一个值：1-上下左右，作用于四边
- 两个值：1-上下，2-左右
- 三个值：1-上，2-左右，3-下
- 四个值：1-上，2-右，3-下，4-左

3. border-color

- 同上

4. border-width

- 同上

5. border-top

- border-top-color

- border-top-style
- border-top-width

6. border-right

- 同上

7. border-bottom

- 同上

8. border-left

- 同上

9. 圆角边框

- border-radius
- border-top-left-radius
- border-top-right-radius
- border-bottom-left-radius
- border-bottom-left-radius

```
<head>
    <meta charset="UTF-8">
    <title>边框属性</title>
    <style>
        #input_1{
            border: double;
            border-style: solid double dotted hidden;
            border-color: red green blue gold;
            border-width: 10px;
        }
        #input_2{
            border-top: 10px solid red;
            border-right: 10px double green;
            border-bottom: 10px dotted blue;
            border-left: 3px dashed gold;
        }
        #input_3{
            border-radius: 5px;
        }
        #input_4{
            border-top-left-radius: 1px;
            border-top-right-radius: 5px;
            border-bottom-left-radius: 5px;
            border-bottom-right-radius: 1px;
        }
    </style>
</head>
<body>
    用户名：<input id="input_1" type="text"/><br/><br/>
    密码：<input id="input_2" type="password"/><br/><br/>
    手机号：<input id="input_3" type="text"/><br/><br/>
    家庭住址：<input id="input_4" type="text"/><br/><br/>
</body>
```

## （四）布局样式

1. 第一步：设置元素在容器块中的定位方式—position

- position：
- static：流定位/默认的定位
- relative：相对定位—相对于自己的定位，设置完成后原来的位置进行保留
- fixed：相对于浏览器窗体进行定位，应用于广告
- absolute：绝对定位，最常用
  - 在参照物里设置position,除了static其他的都可以，建议使用relative
  - 如果参照物不唯一，采用就近原则

```
<head>
    <meta charset="UTF-8">
    <title>absolute--绝对定位</title>son
    <style>
        #grandfather{
            width: 500px;
            height: 500px;
            background-color: yellow;
            position: relative;
        }
        #father{
            width: 300px;
            height: 300px;
            background-color: blue;
            position: relative;
        }
        #son{
            width: 100px;
            height: 100px;
            background-color: red;
            position: absolute;
            right: 20px;
        }
    </style>
</head>
<body>
    <div id="grandfather">
        <div id="father">
            <div id="son"></div>
        </div>
    </div>
</body>
```

2. 第二步：设置定位元素对容器边界的偏移量

- top
- bottom
- left
- right

3. z-index（如果有重叠，设置显示的优先级）

- 使用index前提必须有position
- z-index值越大优先级越高越靠上，越小优先级越低越靠下
- z-index默认值为0

```
<head>
    <meta charset="UTF-8">
    <title>z_index</title>
    <style>
        #one{
            width: 100px;
            height: 100px;
            background-color: yellow;
        }
        #two{
            width: 100px;
            height: 100px;
            background-color: red;
            position: relative;
            top: -90px;
            /*设置z-index，值越小越靠下*/
            z-index: -1;
        }
    </style>
</head>
<body>
    <div id="one"></div>
    <div id="two"></div>
</body>
```

## （五）光标形状

1. cursor

- pointer（手形）
- help（问号）
- text（输入符号）

```
<head>
    <meta charset="UTF-8">
    <title>列表相关属性</title>
    <style>
        div{
            width: 200px;
            height: 200px;
            background-color: blue;
            cursor: help;
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

### （六）块级元素缩放比例

- transform: scale(1.03);

### （七）设置纯色区域透明程度

- opacity：0.5;

## 五、其它样式

### （一）表格

- 设置相邻单元格的边框处理样式：border-collapse
  - separate边框分开
  - collapse边框合并
- 设置相邻单元格边框的间距：border-spacing
- 设置表格标题的位置：mcaption-side
- 设置空单元格是否显示边框：empty-cells
  - hide

```
<head>
    <meta charset="UTF-8">
    <title>表格相关属性</title>
</head>
<body>
    <table border="1px" style="border-collapse:separate; border-spacing: 20px; caption-
side: bottom; empty-cells: hide">
        <caption>标题</caption>
        <tr>
            <td>1-1</td>
            <td>1-2</td>
            <td>1-3</td>
        </tr>
        <tr>
            <td>2-1</td>
            <td></td>
            <td>2-3</td>
        </tr>
        <tr>
            <td>3-1</td>
            <td>3-2</td>
            <td>3-3</td>
        </tr>
    </table>
</body>
```

### （二）列表

- 设置列表标记：list-style-type
- 设置图像列表标记：list-style-image
- list-style-position：设置标记的位置
  - inside(通常情况下，写这个)

    - outside
- 去掉列表前的标记（序号/实心圆点） `list-style: none;`

```
<head>
    <meta charset="UTF-8">
    <title>列表相关属性</title>
    <style>
        ol>li{
            list-style-type: circle;
            list-style-image: url("1.png");
            list-style-position: inherit;
        }
        ul>li{
            /*去掉列表的标记点*/
            list-style: none;
        }
        div{
            width: 200px;
            height: 200px;
            background-color: blue;
            cursor: help;
        }
    </style>
</head>
<body>
    <ol>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
    </ol>

    <l>
        <li>列表项</li>
        <li>列表项</li>
        <li>列表项</li>
    </l>
    <div></div>
</body>
```

# 六、效果

## （一）过渡

1. 什么是过渡：两种状态转换中间的效果
2. 过渡涉及到的属性

- 设置过度开始的延时：transition-delay
- 设置过度持续的时间：transition-duration
- 设置过度的参与属性：transition-property
- 设置过度的速率：transition-timing-function
- 简写属性：transition

3. 创建反向过度

- 直接复制hover设置的内容，到原样式中

```
<head>
    <meta charset="UTF-8">
    <title>过渡</title>
    <style>
        div{
            width: 100px;
            height: 100px;
            background-color: red;

            /*设置延时显示时间*/
            transition-delay: 500ms;
            /*设置过渡持续时间*/
            transition-duration: 3000ms;
            /*设置过渡参与的属性*/
            transition-property: width, height, border-radius, background-color;
            /*设置过渡速率*/
            transition-timing-function: linear;
        }
        div:hover{
            width: 100px;
            height: 100px;
            border-radius: 50px;
            background-color: blue;

            /*设置延时显示时间*/
            transition-delay: 500ms;
            /*设置过渡持续时间*/
            transition-duration: 3000ms;
            /*设置过渡参与的属性*/
            transition-property: width, height, border-radius, background-color;
            /*设置过渡速率*/
            transition-timing-function: linear;
        }
    </style>
</head>
<body>
    <div></div>
</body>
```

### （二）列表快

1. figure img（填入图片） figcaption（写文字部分）

### （三）最大最小尺寸

- 设置最小尺寸：min-width
- 设置最大尺寸：max-width

## 七、实践案例

### （一）网站骨架搭建

1. 网页导航栏图标添加

```
<!--语句一-->
<link rel="shortcut icon" href="baidu.png" type="image/x-icon">
<!--语句二-->
<link rel="icon" href="baidu.png" type="image/x-icon">
```

- 图片也行，icon图标文件也可以，甚至是gif也支持
- 两个语句，只有第一行是必须的，因为“shortcut icon”字符串将被多数遵守标准的浏览器识别为列出可能的关键词（“shortcut”将被忽略，而仅适用“icon”）；而Internet Explorer将会把它作为一个单独的名称（“shortcut icon”）。这样做的结果是所有浏览器都可以理解此代码。只有当希望为新浏览器提供另一种备用图像（例如动画GIF）时，才有必要添加第二行。

2. 搭建时注意

- 把网站层次理清楚
- 使用div进行骨架搭建

> 参考：0.baidu框架搭建.html

## （二）导航栏的制作

- 使用无序列表（ul）

> 参考：1.导航栏的制作.html

## （三）列表块的制作

- eg：天猫的每个商品块
- figure
  - img
  - figcaption（文字信息）

```
<figure class="father">
    <img src="img/block_1.png">
    <figcaption class="son">
        <p class="p_title">大数据</p>
        <p class="p_content">中质信联以电梯安全监管为切入点，以电梯云大数据为支撑，为各单位、社会大众提供大数据服务，实现数据共享。</p>
    </figcaption>
</figure>
```

## （四）最小尺寸，最大尺寸

- min-width
- max-width

> 参考：3.最大最小尺寸.html

## （五）图像遮罩层的制作

- 用div包裹图片
- 图片同级建立一个div当作遮罩层
- 移动同级的div到图片上
- 设置div透明度

> 参考：6.遮罩层的制作

## 第三节：JavaScript

## 第四节：JQuery

# 第二章 Oracle数据库

## 第一节：Oracle安装及使用

### 一、Oracle简介

1. 数据库 Database，简称DB
2. 数据库管理系统 DBMS
3. 关系型数据库管理系统 RDBMS eg: MySQL、Oracle、SQL Sever
4. 主要的数据
    - Mysql 属于Oracle，免费的
    - Oracle 来源与oracle公司
    - DB2 来源于IBM（和IBM相关软件一起）
    - SyBase
    - Sql Sever 微软(MS)
    - Access 微软(MS)
5. oracle公司业务
    - 数据库产品（eg:银行）
    - 认证（培训+考证）OCP OCM
    - 各行各业解决方案
6. Oracle数据库安装文件获取方式 免费下载,oracle是授权方式。（本身不要钱，服务要钱）

### 二、Oracle安装注意事项

1. 在官网下载，根据自己电脑位数下载
2. 必须具备
    - 安装文件所在目录和目标路径不能有汉字
    - 计算机主机名奴能有汉字
    - 计算机用户名不能有汉字
    - 计算机时间 （2039年以前）
    - 防火墙关闭
3. 安装成功后注意事项 最后不要清理文件，防止意外删除Oracle安装的配置文件

### 三、Oracle基础

#### （一）Oracle服务

【Oracle 11g数据库的服务】

1. —>打开服务器：Window+R 输入:`services.msc` 或者：我的电脑->管理->服务和应用程序;

2. 有两个服务必须开启：一个数据库服务`OracleServiceORCL`-Oracle核心服务，一个是监听器服务(OracleOraDb11g_home1TNSListener);
3. 当数据库连接不上的时候，要核实一下两个服务是否打开;
4. 监听服务无法启动，我们需要重建监听:使用Net Configuration Assistant(网络配置助手);
5. 没有ORCL服务或者已有的库不能用了再或者需要新建一个库：使用Database Configuration Assistant(数据库配置助手);

### （二）目录结构

1. product
2. oradata->orcl数据库对应目录

- ctl:Oracle的控制文件
- dbf:Oracle的数据文件
- log:Oracle日志文件

### （三）默认用户

1. sys:角色-sysdba(管理员)
2. system:角色-sysdba
3. scott:角色(一般用户)

### （四）sqlplus

1. oracle客户端（Dos窗口）：
    - 进入数据库:`sqlplus`
    - 普通用户:`scott`
    - 管理员登陆:`sys /as sysdba` `system /as sysdba`

## 四、安装Oracle客户端

- C/S结构（Client/Server）
- SQL Developer 客户端（使用这个即可）
- PL/SQL Developer 第三方厂家开发的客户端（付费、商业版）

# 第二节： 数据库设计

## 一、数据库设计准备

1. 工具：PowerDesigner/ERWin
2. 安装PD

## 二、数据库设计步骤

1. 数据需求分析
2. 逻辑设计(ER图) CDM:概念数据模型
    - 实体:Entity
        - 主键：Primary Key(非空且唯一)
    - 属性：Attribute
    - 实体关系:Entity Relationship
        1. 一对一（1:1）

2. 一对多(1:N)
3. 多对多(M:N)
4. 强制关系与非强制关系

- 逻辑设计步骤

1. 根据需求确定实体:一定找待开发系统相关实体（实体都是名词）；
2. 根据实体确定实体属性：属性类型、长度、非空约束、主键约束；
3. 确定实体之间的关系

- 属性和实体名字注意事项：

1. 不要使用Oracle关键字（Oracle不区分大小写） eg:user,uid,usid，order,desc,from,to,level,group都不能取

3. 物理设计 （表结构）PDM：物理数据模型--->关联到某个数据库

- 表（table）：用于存放数据(ER图中的多对多的关系，再生成物理模型时会被分解两个一对多关系，产生了中间表)。
  - 一行数据-->记录
  - 一列数据（属性）-->字段
  - 五种约束
    1. 非空约束(NOT NULL)：数据不能为空
    2. 唯一约束(UNIQUE)：数据不能重复，Oracle会自动为唯一性约束的字段创建相应的唯一性索引。
    3. 主键约束(PRIMARY KEY):**非空且唯一**，只允许一个主键，主键可以是单个字段或多字段的组合（联合主键），Oracle会自动为主键字段创建对应的唯一性索引。
       - 联合主键：当两个数据表形成的是多对多的关系，那么需要通过两个数据表的主键来组成联合主键，就可以确定每个数据表的其中一条记录了
         - 复合主键：在一个数据表中通过多个字段作为主键来确定一条记录，那么，多个字段组成的就是复合主键
    4. 外键约束(FOREIGN KEY)：建立和其他表的联系，取值可以为空（非强制关系），且外键这一列一定出现在（一对多关系中）多的那一侧。
    5. 检查(CHECK)：
  - VarChar与Char区别：
    1. 定长与变长
    2. char效率高

4. 优化：规范化与反规范化

- 范式（共六个）：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式(4NF）和第五范式（5NF，又称完美范式）
- 什么是范式：用来优化数据数据存储方式的规范；
- 工程上最高可以达到3NF；
- 用友软件的面试题：数据库的三范式分别解决什么问题？ 数据库范式 详解数据库的第一范式、第二范式、第三范式、BCNF范式
  1. 第一范式：符合1NF的关系中的每个属性都不可再分
  2. 第二范式：消除非主属性对码的部分函数依赖
  3. 第三范式：消除非主属性对码的传递函数依赖
  4. BCNF范式：消除主属性对于码的部分与传递函数依赖
- 为什么要规范化？答案：消除冗余数据；
- Oracle（分为两部分）开发+管理----开发容易管理难；

5. 数据库端的开发

# 第三节：SQL语句

## 一、DDL语句

> DDL(Data Definition Language)数据定义语言：操作表结构

### （一）如何创建数据库

```
CREAT DATABASE student;
```

### （二）如何删除数据库

```
DROP DATABASE student;
```

### （三）如何创建表

```
CREATE TABLE T_STUDENT(
    stu_id VARCHAR2(20),
    stu_name VARCHAR2(20),
    sex VARCHAR2(2),
    birthday DATE,
    score NUMBER,
    mobile VARCHAR2(11),
    stu_class_id VARCHAR2(10)
);
```

### （四）什么是约束?（点击）

- 列级约束：包含在列定义中，是对某一个特定列的约束，直接跟在列的其它定义之后，用空格分开，不必指定列名
    - 主键、外键、唯一、检查、**缺省、非空**
- 表级约束：与列定义相互独立，不包含在列定义中（通常用于对多个列一起进行约束，必须指出要约束列 的名称）
    - 主键、外键、唯一、检查

1. 主键约束（PRIMARY KEY）：非空且唯一
2. 唯一约束（UNIQUE）：不可重复，可以为空(为空的可重复)
3. 检查约束（CHECK）：对该列数据的范围、格式的限制（如：年龄、性别等）
4. 外键约束（FOREIGN KEY）：需要建立两表间的关系并引用主表的列
5. 非空约束（NOT NULL）：数据不能为空
6. 默认/缺省约束（DEFAULT）:该数据的默认值

```
/*行内约束*/
CREATE TABLE T_STUDENT(
    stu_id VARCHAR2(20) PRIMARY KEY,/*主键约束*/
    stu_name VARCHAR2(20) NOT NULL,/*非空约束*/
    sex VARCHAR2(2) DEFAULT ('男'),/*默认/缺省约束*/
    birthday DATE NOT NULL,/*非空约束*/
    score NUMBER NOT NULL CHECK(core >= 2.0 AND core <= 4.0),/*检查约束*/
    mobile VARCHAR2(11) UNIQUE,/*唯一约束*/
    stu_class_id VARCHAR2(10) REFERENCES T_Class(class_id)/*外键约束*/
);
```

```
/*行外约束*/
CREATE TABLE T_STUDENT(
    stu_id VARCHAR2(20),
    stu_name VARCHAR2(20) NOT NULL,/*非空约束*/
    sex VARCHAR2(2) DEFAULT ('男'),/*默认/缺省约束*/
    birthday DATE NOT NULL,
    score NUMBER NOT NULL,
    mobile VARCHAR2(11),
    stu_class_id VARCHAR2(10),

    CONSTRAINT PK_STUDENT PRIMARY KEY (stu_id),/*主键约束*/
    CONSTRAINT check_score CHECK (score >= 2.0 AND score <= 4.0),/*检查约束*/
    CONSTRAINT unique_mobile UNIQUE (mobile),/*唯一约束*/
    CONSTRAINT FK_STUDENT FOREIGN KEY (stu_class_id) REFERENCES T_Class(class_id)/*外键约束*/
);
```

```
/*增加约束(在表建立后)*/
ALTER TABLE  T_STUDENT ADD CONSTRAINT FK_STUDENT FOREIGN KEY(stu_class_id ) REFERENCES
T_Class(class_id);
```

```
/*禁用或启用约束*/
ALTER TABLE 表名  DISABLE|ENABLE  CONSTRAINT 约束名;

ALTER TABLE T_STUDENT DISABLE CONSTRAINT FK_STUDENT;--禁用约束
ALTER TABLE T_STUDENT ENABLE CONSTRAINT FK_STUDENT;--启用约束
```

### (五) 如何删除表

```
DROP TABLE t_user;/*连表和数据全部删除*/
TRUNCATE TABLE t_user;/*保留表结构，数据全删除，不可回退*/
```

### (六) 如何修改表 (表结构)

```
DESC T_USER; /*显示表结构*/
RENAME T_USER TO T_USER2;/*修改表名-第二种方式*/
ALTER TABLE T_USER2 RENAME TO T_USER;/*修改表名-第二种方式*/

ALTER TABLE T_USER ADD mobile VARCHAR2(11);/*添加字段*/
ALTER TABLE T_USER DROP COLUMN mobile;/*删除字段*/
ALTER TABLE T_USER MODIFY userid VARCHAR(20);/*修改字段类型*/
ALTER TABLE T_USER RENAME COLUMN userid TO user_id;/*修改字段名*/
```

### （七）如何修改表内容

```
INSERT INTO T_USER VALUES (1,'zhangsan',22,'男','zhangsan@163.com');-- 默认插入(按照定义顺序依次填写)
INSERT INTO T_USER(id,name,age,sex,email) VALUES (1,'zhangsan',22,'男','zhangsan@163.com');-- 通过给定字段传入
UPDATE T_USER SET name='lisi' WHERE id = 1;-- 表的更新操作，更改指定id的属性值
DELETE FROM T_USER WHERE id = 1;-- 删除id为1的数据
SELECT * FROM T_USER;-- 查看表内容
```

### （八）提交数据

```
COMMIT;-- 向数据库提交(INSERT INTO 后要使用COMMIT提交事务，否则数据不会被提交到表里面去)
```

### （九）获取表信息

1. 获取表：

```
select table_name from user_tables; /*当前用户的表*/
select table_name from all_tables; /*所有用户的表*/
select table_name from dba_tables; /*包括系统表*/
```

2. 获取表字段：

```
select * from user_tab_columns where Table_Name='用户表';
select * from all_tab_columns where Table_Name='用户表';
```

3. 获取表注释：

```
select * from user_tab_comments;
```

4. 获取字段注释

```
select * from user_col_comments;
```

## 二、DQL语句

> DQL (Data Query Language)数据查询语言
>
> - SQL92、SQL99标准
> - DQL:Data Query Language----数据查询语言

> - SQL:Structured Query Language---结构化查询语言

## （一）查询表内容

```
SELECT * FROM classes;-- 查询表所有列（FROM比SELECT先执行）
SELECT class_id FROM classes;-- 查询部分列（投影）
```

面试题：SELECT 和 FROM 谁先执行？（FROM）

## （二）使用别名(字段名、表名)

字段名和表名+空格+别名,也可使用AS

```
/*表的别名*/
SELECT  c.class_id,c.class_name FROM classes c;-- 表后直接跟上别名
/*字段的别名*/
SELECT class_id 学生编号, class_name 学生姓名 FROM classes;-- 直接在字段后面加别名
SELECT class_id "学生编号", class_name "学生姓名" FROM classes;-- 别名使用双引号
SELECT class_id AS 学生编号, class_name AS 学生姓名 FROM classes;-- AS 加 别名
SELECT class_id AS "学生编号", class_name AS "学生姓名" FROM classes;-- AS + 引号别名
```

## （三）去掉重复行（DISTINCT）

重复：指定的字段要完全相同

```
-- 去除班级名相同的数据并进行显示
SELECT DISTINCT class_name FROM classes;
-- 去除班级名和编号都相同的数据并进行显示
SELECT DISTINCT class_name,class_id FROM classes;
```

## （四）连接字段(||或者CONCAT)

```
-- 将学生班级编号和班级名在一列中进行显示
SELECT  class_id||class_name FROM classes;-- 使用“||”
SELECT CONCAT(class_id,class_name) AS "班级信息" FROM classes; -- 使用“CONCAT”
```

## （五）排序(ORDER BY)

```
-- 根据班级编号大小进行排序
SELECT  class_id,class_name FROM classes ORDER BY class_id;-- 默认升序
SELECT  class_id,class_name FROM classes ORDER BY class_id ASC;-- 升序(ASC)
SELECT  class_id,class_name FROM classes ORDER BY class_id DESC;-- 降序(DESC)
SELECT  class_id AS cid,class_name FROM classes ORDER BY cid DESC;-- 利用字段别名排序
```

```
-- 按多字段排序(先按照第一个字段排序，如果相同，再按第二个字段排序)
SELECT classid, classname FROM classes ORDER BY classid,classname DESC;
--按非选择列排序
SELECT   classname    FROM classes    ORDER BY classid  DESC;
```

## （六）限定查询(WHERE)

1. 比较运算符 =， >，<，<>，>=，<=，!=

```
SELECT * FROM student WHERE sex = '女';
SELECT * FROM student WHERE sex != '女';
SELECT * FROM student WHERE score >= 3.0 ;
SELECT * FROM student WHERE score <= 3.0 ;
SELECT * FROM student WHERE stu_class_id != '10';
-- 查询班号不为"10"的学生信息(<>相当于!=)
SELECT * FROM student WHERE stu_class_id <> '10';
```

2. AND

```
-- 查询学分大于3.0的女学生的信息
SELECT * FROM student WHERE score>3.0 AND sex = '女';
```

3. OR

```
-- 查询班号为"10"或者班号为"30"的学生信息
SELECT * FROM student WHERE stu_class_id = '10' OR stu_class_id = '30';
-- 查询班号为"10"的男学生信息或者学分大于3.0的学生信息
SELECT * FROM student WHERE score>3.0 OR Sex = '男' AND stu_class_id = '10';/*AND优先级比OR高*/
```

4. IN / NOT IN

```
--查询班号是"10"、"20"的学生信息。
SELECT * FROM student WHERE stu_class_id IN('10','30');
--查询班号不是"10"、"20"的学生信息。
SELECT * FROM student WHERE stu_class_id NOT IN('10','30');
```

5. BETWEEN AND / NOT BETWEEN AND(含边界值)

```
-- 查询学分在2.5~3.5之中的学生信息
SELECT * FROM student WHERE score BETWEEN 2.5 AND 3.5;
-- 查询学分不在2.5~3.5之中的学生信息
SELECT * FROM student WHERE Score NOT BETWEEN 2.5 AND 3.5;
```

6. IS NULL / NOT IS NULL

```
-- 查询电话非空的学生信息
SELECT * FROM student WHERE mobile IS NULL;
-- 查询出生日期不为空的学生信息
SELECT * FROM student WHERE birthday IS NOT NULL;
```

6. LIKE(模糊查询)

```
/*LIKE*/
-- 查询所有姓"孙"的学生信息
SELECT * FROM student WHERE stu_name LIKE '孙%';
-- 查询所有名字中含有"辉"的学生信息
SELECT * FROM student WHERE stu_name LIKE '%辉%';
-- 查询所有名字以"五"结尾的学生信息
SELECT * FROM student WHERE stu_name LIKE '%五';
-- 查询所有名字中含有"辉"且前后各只有一个字符的学生信息（"_": 表示单个字符）
```

```
SELECT * FROM student WHERE stu_name LIKE '_辉_';
-- 查询所有手机号不为空的学生信息
SELECT * FROM student WHERE mobile LIKE '%%';--任意值（非空）
-- 查询所有名字中含有"_"的学生信息(ESCAPE：自定义转义字符)
SELECT * FROM student WHERE stu_name LIKE '%\_%' ESCAPE '\';
/*NOT LIKE*/
--查询所有不是姓"张"的学生信息
SELECT * FROM student WHERE stu_name NOT LIKE '张%';
```

面试题：

```
-- 表 book（bookname，pub）以下两个SQL语句有何区别?
SELECT * FROM book;
SELECT * FROM book WHERE bookname LIKE '%%' AND pub LIKE '%%'
/*
    1.通过第一种方式会查询所有数据
    2.通过第二种方式查询，结果中不包含空的数据
*/
CREATE TABLE book(
    bookname VARCHAR2(20),
    pub VARCHAR(20)
);
INSERT INTO book VALUES('操作系统','');-- pub为空
INSERT INTO book VALUES('编译原理','');
INSERT INTO book VALUES('','电子工业出版社');-- bookname为空
INSERT INTO book VALUES('','人民邮电出版社');
INSERT INTO book VALUES('软件工程','清华大学出版社');--都不为空
INSERT INTO book VALUES('Java程序设计','东南大学出版社');
SELECT * FROM book;
SELECT * FROM book WHERE bookname LIKE '%%' AND pub LIKE '%%';
```

8. 对查询结果分组 GROUP BY 和 HAVING

- 分组函数（Oracle内置函数）
    - max 求一列数据的最大值，适用任何数据类型
    - min 求一列数据的最小值，适用任何数据类型
    - sum 求一列数据的总和，只能用于数字
    - avg 求一列数据的品平均值，只能用于数字
    - count 统计表数据总行数

```
    面试题:COUNT(字段)和COUNT(*)区别
    COUNT(字段)：不统计空数据
    COUNT(*):统计所有的数据
```

```
-- SELECT完整语法
SELECT column, group_function
FROM table  WHERE condition
GROUP BY group_by_expression -- 按什么字段分组
HAVING group_condition -- 对分组后的结果进行限制
ORDER BY column;   -- 总是在最后
```

```
/*单列分组*/
--任务：显示每个部门的平均工资
SELECT deptno, AVG(sal) avgsal FROM emp GROUP BY deptno;--按 deptno分组
/*多列分组*/
SELECT deptno,job,AVG(sal) avgsal,MAX(sal) maxsal FROM emp GROUP BY deptno,job; --按deptno和job一起分组
/*使用HAVING*/
-- 任务：显示平均工资小于2500的各部门号、平均工资、最低工资
SELECT empno,AVG(sal),MIN(sal) FROM emp GROUP BY empno HAVING AVG(sal)<2500;
-- 任务：显示各部门部门编号,平均工资，按平均工资排序。对分组结果排序
SELECT deptno,ROUND(AVG(sal),3) FROM emp GROUP BY deptno ORDER BY AVG(sal);--ASC,DESC
```

```
面试题：SELECT各子句执行次序
执行先后次序： from -> where -> group by -> having -> select -> order by
eg:SELECT ename,sal salary FROM emp WHERE salary < 2000; --该语句返回0条记录还是可能有多条？答案：报错，不返回值。
```

9. 伪列rownum、rowid

```
任务：查询学生表的前3行数据。
SELECT * FROM emp WHERE rownum < 3;
任务：查询学生表的第2至第4行数据(使用 minus)
SELECT * FROM emp WHERE rownum <= 4 MINUS SELECT * FROM emp WHERE rownum  <= 1;
select * from (select t.*, rownum rn from emp t) where rn >= 2 AND rn <= 4;
任务：理解rownum、rowid
SELECT rownum,rowid FROM student;
```

- ROWID：ROWID是ORACLE中的一个重要的概念。用于定位数据库中一条记录的一个相对唯一地址值。通常情况下，该值在该行数据插入到数据库表时即被确定且唯一。ROWID它是一个伪列，它并不实际存在于表中。它是ORACLE在读取表中数据行时，根据每一行数据的物理地址信息编码而成的一个伪列。所以根据一行数据的ROWID能找到一行数据的物理地址信息。从而快速地定位到数据行。数据库的大多数操作都是通过ROWID来完成的，而且使用ROWID来进行单记录定位速度是最快的。
- rowid是数据的详细地址，利用rowid可以帮助oracle快速定位某行数据的具体位置。rowid可以分为两种，普通表中的rowid是物理rowid，索引组织表中的rowid是逻辑rowid。
- 使用describle(或简写为desc)命令查看表结构时，输出结果中是不能看到rowid这里一列的，这是因为这一列只在数据库内部使用。

# 三、DML语句

> DML(Data Manipulation Language)数据操纵语言：操作表中的数据 包含：INSERT,UPDATE,DELETE

## (一) INSERT INTO

```
INSERT INTO dept VALUES(20,'K','L'); -- 给所有字段设置值
INSERT INTO dept(deptno,dname) VALUES(20,'K'); --设置指定字段值
INSERT INTO dept VALUES(60,'MARKET',DEFAULT); -- 使用默认值
INSERT INTO employee (empno,ename,sal,deptno) SELECT empno,ename,sal,deptno FROM emp WHERE
deptno=20; -- 在INSERT中使用子查询
-- 使用多表插入数据（不常用）
INSERT ALL
WHEN deptno=10 THEN INTO dept10
WHEN deptno=20 THEN INTO dept20
WHEN deptno=30 THEN INTO dept30
WHEN job='CLERK' THEN INTO clerk
ELSE INTO other
SELECT * FROM emp;
COMMIT;--提交事务
```

## （二）UPDATE

```
UPDATE emp SET sal=2560 WHERE ename='SCOTT';
UPDATE emp SET sal=sal*1.5,comm=sal*0.1 WHERE deptno=20;
UPDATE emp SET hiredate=TO_DATE('2007-12-25','yyyy-mm-dd') WHERE empno=7788;
UPDATE emp SET (job,sal,comm)=(SELECT job,sal,comm FROM emp WHERE ename='SMITH') WHERE
ename='SCOTT';
UPDATE employee SET deptno=(SELECT deptno FROM emp WHERE empno=7788) WHERE job=(SELECT job FROM
emp WHERE empno=7788); --基于一张表修改另一张表
COMMIT;--提交事务
```

## （三）DELETE

```
DELETE FROM emp WHERE ename='SMITH'; --FROM可以省略
DELETE FROM emp;
DELETE FROM emp WHERE deptno=(SELECT deptno FROM dept WHERE dname='SALES');-- DELETE中使用子查询
COMMIT;--提交事务
```

## （四）事务 Transaction（面试重点）

1. 事务使用场景 银行转账。从A账户向B账户转账100元，将会执行几个SQL语句？ 如果其中一个SQL语句执行失败，最终结果会如何?

```
UPDATE account SET balance=balance-100 WHERE account='A';
UPDATE account SET balance=banlance+100 WHERE account='B';--执行失败，最终结果呢?
```

```
UPDATE account SET balance=balance-100 WHERE account='A';
UPDATE account SET balance=banlance+100 WHERE account='B';--执行失败
ROLLBACK;--回滚事务，回退事务，撤销第一个句子的结果
```

2. 事务是什么？ 事务（Transaction）就是一组SQL语句，这组SQL语句是一个逻辑工作单元，也可以认为事务是一组不可分 割的SQL语句。全部SQL执行成功，提交事务；只要其中一个SQL语句执行失败，回退事务，就可认为“这件事情没做”

```
--提交事务 COMMIT;
--回滚或回退事务 ROLLBACK;
```

3. 事务属性有哪些？ 4个
    - 原子性（Atomicity）：事务中的所有SQL语句的执行是不可分割的
    - 一致性（Consistency）：确保数据库的状态是一致的，事务开始时，数据库的状态是一致的；在事务结束时候，数据库的状态也是一致的
    - 隔离性（Isolation）：多个事务可以独立运行，彼此不产生影响
    - 持久性：（Durability）：一旦事务提交，数据库的变化就会永久地保留下来

### （五）索引 INDEX

参考链接

### （六）序列

参考连接

1. nextval 写一个序列值
2. currval 序列当前值

### （七）虚表 dual（常量查询）

## 四、多表连接查询

1. 多表连接查询 (面试题)
2. 视图 (面试题)
3. 子查询 (面试题)
4. CASE表达式
5. 空值函数

### （一）Oracle 多表连接查询

> 表的三种连接方式：内链接、外连接、交叉连接(两张表关联的载体就是主外键)

1. 内链接
    - 等值连接（在连接条件中使用“=”）
    - 不等值连接（在连接条件中不适用“=”）
    - 自然连接：(Natural join)是一种特殊的等值连接，它要求两个关系中进行比较的分量必须是相同的属性组，并且在结果中把重复的属性列去掉，而等值连接并不去掉重复的属性列。
    - 自连接

```
--<等值连接>
-- 显示学生学号、姓名和班级名称
-- 1.旧写法
SELECT stu_id, stu_name, class_name FROM t_student a, t_class b WHERE a.stu_class_id =
b.class_id;
-- 2.新写法 INNER JOIN ...ON (条件);
SELECT stu_id, stu_name, class_name FROM t_student a INNER JOIN t_class b ON a.stu_class_id =
b.class_id;
-- 显示学生学号、姓名和班级名称、班号（如果两表相关联字段的字段名相同，字段要指明表名）
SELECT stu_id ,stu_name,class_name, class_id FROM t_student a INNER JOIN t_class b ON A.class_id
= b.class_id;
--<非等值连接>
SELECT e.ename,e.sal,s.grade FROM emp e, salgrade s WHERE e.sal BETWEEN s.losal AND s.hisal;
--<自然连接>
SELECT * FROM R INNER JOIN S ON S.b = R.b;-- 不去掉重复的属性列
```

```
SELECT * FROM R Natural INNER JOIN S;-- 把重复的属性列去掉
--<自连接>
-- 显示员工姓名以及上级领导姓名
SELECT w.ename, m.ename FROM emp w, emp m WHERE w.mgr = m.empno;
```

2. 外连接（外键列取值有空的场景）
    - 左外连接（left outer join）
    - 右外连接（right outer join）
    - 全连接（full outer join）

```
-- 显示所有学生的学号、姓名和班级名称
-- <1.左连接left outer join> 查询结果除了返回包含连接条件的行，还包含左表（a）中不满足连接条件的行，其中不满足连接条件的行中b表的字段值将被置为空
SELECT stu_id ,stu_name,class_name, STU_CLASS_ID FROM t_student a LEFT OUTER JOIN t_class b ON a.stu_class_id = b.class_id;
-- <2.右连接right outer join> 查询结果除了返回包含连接条件的行，还包含右表（b）中不满足连接条件的行，其中不满足连接条件的行中a表的字段值将被置为空。
SELECT stu_id ,stu_name,class_name, STU_CLASS_ID FROM t_student a RIGHT OUTER JOIN t_class b ON a.stu_class_id = b.class_id;
-- <3.全连接full outer join> 查询结果除了返回包含连接条件的行，还包含左表(a)、右表（b）中不满足连接条件的行，其中不满足连接条件的行中的字段值将被置为空。
SELECT stu_id ,stu_name,class_name, STU_CLASS_ID FROM t_student a FULL OUTER JOIN t_class b ON a.stu_class_id = b.class_id;
```

3. 交叉连接

```
SELECT stu_id ,stu_name,class_name, STU_CLASS_ID FROM t_student,t_class ; -- 默认就是交叉连接
SELECT stu_id ,stu_name,class_name, STU_CLASS_ID FROM t_student a CROSS JOIN t_class b;
```

4. 多张表查询

```
-- 旧写法
SELECT stu_id ,stu_name,class_name,major_name FROM t_student a,t_class b,t_major c WHERE a.STU_CLASS_ID = B.CLASS_ID AND b.class_major_id = c.major_id;
-- 新写法
SELECT stu_id ,stu_name,class_name,major_name FROM t_student a INNER JOIN t_class b ON a.STU_CLASS_ID = b.class_id INNER JOIN t_major c ON b.class_major_id = c.major_id;
```

## （二）视图：View

1. 试图是什么？视图（审视数据的窗口）,是一个虚拟表，本质是SELECT语句，基于已有的表（基表）建立，有一张或者多张表构成

> 面试题：视图中是否存储数据？不可以 面试题：能否对视图进行增删改操作？一般可以，设置了只读的视图不可以
> 面试题：使用视图的目的是什么？1.可以隐藏一些数据（保护数据）2.可使复杂的查询易于管理和使用（简化操作）

2. 如何创建视图？

```
CREATE [OR REPLACE] VIEW view_name AS
SELECT查询
[WITH READ ONLY] --只读视图
```

```
-- 创建一个视图，仅包含学生学号，姓名，学分绩点，班级名称；从视图中查询学生姓名和班级名称
CREATE VIEW stu_view AS SELECT stu_id ,stu_name,class_name,score FROM t_student INNER JOIN
t_class ON STU_CLASS_ID = class_id;
```

3. 如何使用视图

```
-- 和表的使用一样
SELECT * FROM stu_view;
```

4. 如何删除视图

```
DROP VIEW stu_view;
```

## （三）子查询

1. 什么是子查询？ SELECT语句的嵌套。
2. 何时使用子查询？ 当要显示的数据 在表里并不存在，但可以通过对已有数据的加 工获得，可通过子查询实现。
3. 子查询分类?
    - 单行子查询
    - 多行子查询
    - 多列子查询

```
-- <1.单行子查询>
-- 查询与SCOTT同一部门的员工信息
SELECT ename,sal,deptno FROM emp WHERE deptno=(SELECT deptno FROM emp WHERE ename='SCOTT');
-- <2.多行子查询>
-- 查询与10号部门员工职位相同的员工信息。
SELECT ename,job,sal,deptno FROM emp WHERE job IN (SELECT DISTINCT job FROM emp WHERE deptno=10);
SELECT ename,sal,deptno FROM emp WHERE sal> ALL (SELECT sal FROM emp WHERE deptno=30);-- ALL
SELECT ename,sal,deptno FROM emp WHERE sal> ANY (SELECT sal FROM emp WHERE deptno=30); -- ANY
-- <3.多列子查询>
-- 查询与SMITH所在部门和职务相同的员工信息
SELECT ename,job,sal,deptno FROM emp WHERE (deptno,job)=(SELECT deptno,job FROM emp WHERE
ename='SMITH');
-- 高于部门平均工资的雇员信息
SELECT ename,job,sal FROM emp INNER JOIN (SELECT deptno,avg(sal) avgsal FROM emp GROUP BY deptno)
dept ON emp.deptno=dept.deptno AND sal>dept.avgsal;
```

> 面试题：何为行内视图？和普通视图有何区别？ 行内视图只可用一次，普通视图存放于Oracle中，可随时使用

4. 子查询可以出现在哪里？ 子查询可以出现在4种地方：WHERE子句、FROM子句、DML语句、DDL语句中
5. 子查询编写思路?
    - 仔细分析题目，确定要查询的表及字段（数据）
    - 分析要查询的字段（数据）哪些在表里直接存在，哪些不存在
    - 考虑如何把要显示的数据造出来（通过查询语句获得）
    - 考虑子查询与表的连接点是什么 (通常是主外键、共有字段)
    - 考虑子查询放在什么位置
    - 组合成完整的SQL语句 **提示：工程上，把复杂的子查询做成视图**

## （四）CASE

```
--对10号部门员工工资定级,员工工资大于3000的grade是1，大于2000的grade是2，否则grade是3
SELECT ename,sal, CASE WHEN sal>3000 THEN 1
                       WHEN sal>2000 THEN 2
                       ELSE 3 END grade
FROM emp WHERE deptno=10;
```

## （五）Oracle空值函数 NVL （重要）

```
-- 显示员工姓名和补助（comm）
SELECT ename, comm FROM emp;
-- 显示员工姓名和补助（comm），补助为空NULL的，用0替换
SELECT enamel, NVL(comm,0) FROM emp;
```

## （六）课后任务

- （使用到Oracle自带的表：DEPT、EMP、SALGRADE ）

1. 任务：百度搜索“自然连接”并理解之（下次课随机抽取讲解）
2. 任务：显示学生姓名、班级名称、专业名称。使用3张表：学生表，班级表，专业表
3. 任务：显示班级编号为10的学生姓名、班级名称、专业名称。使用3张表：学生表，班级表，专业表
4. 任务：创建一个视图，包含学生姓名、学分绩点、手机、班号、班级名称、专业名称。从该视图中查询班号为10的学生姓名、班级名称。写出创建视图语句和SQL语句。
5. 任务：搜索Oracle集合操作并学之[UNION|UNION ALL|INTERSECT|MINUS]（下次课随机抽取讲解）。
6. 显示所有雇员的平均工资、总工资、最高工资、最低工资
7. 显示每种岗位的雇员总数、平均工资
8. 显示雇员总数，以及获得补助的雇员数
9. 显示管理者总数
10. 显示雇员工资的最大差额
11. 显示每部门每岗位的平均工资、每个部门的平均工资、每岗位的平均工资
12. 显示部门20的部门名称、以及该部门的所有雇员名字、雇员工资和岗位
13. 显示获得补助的所有雇员名字、补助以及所在部门
14. 显示在DALLAS工作的所有雇员名字、雇员工资以及所在部门
15. 显示雇员SCOTT的管理者名字
16. 查询EMP表和SALGRADE表，显示部门20的雇员名字、工资以及其工资级别
17. 显示部门10的所有雇员名字、部门名称、以及其他部门名称
18. 18. 显示部门10的所有雇员名字、部门名称、以及其他雇员名称
19. 显示部门10的所有雇员名字、部门名称、以及其他部门名称和雇员名称

选做：

1. 显示BLAKE同部门的所有员工，但不显示BLAKE
2. 显示超过平均工资的所有员工名、工资和部门号
3. 显示超过部门平均工资的所有员工名、工资和部门号
4. 显示高于CLERK岗位所有雇员工资的所有雇员名、工资和岗位
5. 显示工资、补助与SCOTT完全一致的所有雇员名、工资和补助

# 五、Oracle函数

## （一）函数分类

1. 内置函数
2. 自定义函数

## （二）常用函数

- 字符函数
- 数值函数
- 日期函数
- 通用函数
- 统计函数

```
--1.1字符函数
/*
UPPER(列|字符串):将字符串的内容全部转为大写
LOWER(列|字符串):将字符串的内容全部转为小写
INITCAP(列|字符串):将字符串的首字母转为大写
LENGTH(列|字符串):求出字符串的长度
SUBSTR(列|字符串，起始索引，长度):截取字符串
REPLACE(列|字符串):字符串替换
TRIM(列|字符串):去掉左右空格
INSTR(列|字符串，要查找的字符串 ):查找字符串中某个字符的索引
*/
SELECT * FROM EMP;
SELECT UPPER(ENAME),ENAME FROM EMP;/*转为大写*/
SELECT LOWER(ENAME),ENAME FROM EMP;/*转为小写*/
SELECT INITCAP(ENAME),ENAME FROM EMP;/*首字母大写*/
SELECT LENGTH(ENAME),ENAME FROM EMP;/*获取字符串的长度*/
SELECT SUBSTR(ENAME,1,3),ENAME FROM EMP;/*截取字符串*/
SELECT REPLACE(ENAME,'S','XXX'),ENAME FROM EMP;/*字符串替换*/
SELECT TRIM(ENAME) FROM EMP;/*去除左右空格*/
SELECT INSTR(ENAME,'E'), ENAME FROM EMP;/*查找字符串中某个字符的索引(找到返回索引，找不到返回0)*/


--1.2数值函数
/*
    ROUND(数字，保留位数)：对小数进行四舍五入，可以指定保留的位数，不指定则会将小数点之后的数字全部进行四舍五入
    MOD(数字，数字):取模
*/
CREATE TABLE T_SALARY(/*创建一个工资表*/
    EID NUMBER PRIMARY KEY,
    ENAME VARCHAR2(20),
    ESALARY NUMBER
);
INSERT INTO T_SALARY(EID,ENAME,ESALARY) VALUES(1,'zhangsan',4536.25546);
INSERT INTO T_SALARY(EID,ENAME,ESALARY) VALUES(2,'lisi',3985.65845);
INSERT INTO T_SALARY(EID,ENAME,ESALARY) VALUES(3,'wangwu',4937.35468);
SELECT * FROM T_SALARY;
/*将员工工资保留三位小数*/
SELECT ROUND(s.ESALARY),s.ESALARY FROM T_SALARY s;
SELECT ROUND(s.ESALARY,3),s.ESALARY FROM T_SALARY s;
/*将员工工资对1000取模（余数）*/
```

```
SELECT MOD(s.ESALARY,1000),s.ESALARY FROM T_SALARY s;


--1.3日期函数
/*
    ADDDATE(日期，数字)：指定的日起加上指定的天数，求出新的日期(MySQL写法)
    日期+数字：指定的日起加上指定的天数，求出新的日期(Oracle写法)
    ADD_MONTHS(日期，数字)：指定的日起加上指定的约月数，求出新的日期(Oracle写法)
    LAST_DAY(日期)：求出指定日期(当月)的最后一天
*/
-- 1.3.1获得对当前系统时间
/*Oracle 写法*/
SELECT SYSDATE FROM DUAL;
SELECT TO_CHAR(SYSDATE,'yyyy-MM-dd HH24:mi:ss') FROM DUAL;
/* MySQL写法*/
SELECT NOW() FROM t_user;

--1.32
SELECT TO_CHAR(e.HIREDATE,'yyyy-mm-dd') FROM EMP e;
/*加一天*/
SELECT adddate(NOW(),1),NOW() FROM t_user;/*MySQL写法*/
SELECT TO_CHAR(SYSDATE + 1,'yyyy-mm-dd hh24-mi-ss') FROM DUAL;
SELECT TO_CHAR(e.HIREDATE + 1,'yyyy-MM-dd HH24:mi:ss'),TO_CHAR(e.HIREDATE,'yyyy-MM-dd
HH24:mi:ss') FROM EMP e;
/*加一个月*/
SELECT TO_CHAR(ADD_MONTHS(SYSDATE,1),'yyyy-mm-dd hh24-mi-ss') FROM DUAL;
SELECT TO_CHAR(ADD_MONTHS(e.HIREDATE,1),'yyyy-MM-dd HH24:mi:ss'),TO_CHAR(e.HIREDATE,'yyyy-MM-dd
HH24:mi:ss') FROM EMP e;
/*加一年*/
SELECT TO_CHAR(ADD_MONTHS(SYSDATE,12),'yyyy-mm-dd hh24-mi-ss') FROM DUAL;
SELECT TO_CHAR(ADD_MONTHS(e.HIREDATE,12),'yyyy-MM-dd HH24:mi:ss'),TO_CHAR(e.HIREDATE,'yyyy-MM-dd
HH24:mi:ss') FROM EMP e;

/*计算当前月的最后一天*/
SELECT TO_CHAR(LAST_DAY(ADD_MONTHS(SYSDATE,-1)),'yyyy-mm-dd') FROM DUAL;


--1.4通用函数
/*
    IFNULL(字段名，默认值)：如果该字段数据为NULL,就是用默认值 (MySQL)
    NVL(col,val):当col为空时取val作为返回值，当col不为空时取col值
    DECODE(字段,判断值1，返回值1，判断值2，返回值2，... ，默认值):多值判断，如果某一列的数据与判断值相同，则使用指定的显示结果输出，如果没有满足条件，则显示默认值
    CASE 列|数值 WHEN 表达式1 THEN 显示结果1 ... ELSE 表达式2 END:用于实现多条判断，在WHERE之后编写条件，而在THEN之后 编写条件满足的显示操作，如果不满足则使用ELSE中的表达式处理
*/
SELECT * FROM EMP;
SELECT NVL(e.COMM,2333) ,e.COMM FROM EMP e;

CREATE TABLE T_USER(/*创建一个用户表*/
    USER_ID NUMBER PRIMARY KEY,
    USER_NAME VARCHAR2(20),
    SEX VARCHAR2(20)
);
INSERT INTO T_USER(USER_ID,USER_NAME,SEX) VALUES(1,'zhangsan',null);
INSERT INTO T_USER(USER_ID,USER_NAME,SEX) VALUES(2,'lisi','男');
```

```
INSERT INTO T_USER(USER_ID,USER_NAME,SEX) VALUES(3,'wangwu','女');
SELECT *FROM T_USER;
/*空值判断*/
SELECT NVL(u.SEX,'中性'),u.SEX FROM T_USER u;
/*CASE判断*/
SELECT CASE WHEN u.SEX = '男' THEN '♂' ELSE '♀' END, u.SEX FROM T_USER u;
/*DECODE多值判断*/
SELECT DECODE(u.SEX,'男','♂','女','♀',null) FROM T_USER u;

--1.5统计函数
/*
    COUNT：求出全部的居记录数(分页的时候会用到)
    SUM：求出总和
    AVG:平均值
    MAX:最大值
    MIN:最小值
*/
SELECT * FROM EMP;
/*查询员工总数*/
SELECT COUNT(1) FROM EMP;
/*求出每个月支出的总和*/
SELECT SUM(SAL)+SUM(COMM) AS SUM FROM EMP;
/*求出员工最高工资*/
SELECT MAX(SAL) FROM EMP;
/*求出员工最低工资*/
SELECT MIN(SAL) FROM EMP;
/*求出员工平均工资*/
SELECT AVG(SAL) FROM EMP;
/*都显示出来*/
SELECT MAX(SAL) AS "最高工资", MIN(SAL) AS "最低工资", ROUND(AVG(SAL),3) AS "平均工资" FROM EMP;
```

## （三）一般函数

```
SELECT * FROM emp;

-- 1.ASCII
-- 返回指定的字符饭回来的十进制数
SELECT ascii('A'),ascii('a'),ascii('0'),ascii(' ') FROM dual;

-- 2.CHR
-- 给出整数返回对应的字符
SELECT chr(65),chr(97),chr(48),chr(32) FROM dual;

-- 3.CONCAT
-- 连接两个字符串;
  SELECT concat('010-','88888888')||'转110'  电话 FROM dual;

-- 4.INITCAP
-- 返回字符串并将字符串的第一个字母变为大写;
  SELECT initcap('smith') upp FROM dual;

-- 5.INSTR(C1,C2,I,J)
-- 在一个字符串中搜索指定的字符,返回发现指定的字符的位置;
  SELECT instr('oracle traning','ra',1,2) instring FROM dual;
```

```sql
-- 6.LENGTH *
-- 返回字符串的长度
SELECT  ename, length(ename)  FROM emp;

-- 7.LOWER
-- 返回字符串,并将所有的字符小写
SELECT lower('AaBbCcDd')   AaBbCcDd FROM dual;
SELECT deptno FROM emp WHERE lower(ename)='scott';

-- 8.UPPER
-- 返回字符串,并将所有的字符大写
  SELECT upper('AaBbCcDd') upper FROM dual;

-- 9.RPAD和LPAD
-- RPAD  在列的右边粘贴字符
-- LPAD  在列的左边粘贴字符
-- 不够字符则用*来填满
SELECT rpad('gao',5,'*') FROM dual;
SELECT lpad('gao',5,'*') FROM dual;
SELECT lpad(rpad('gao',10,'*'),17,'*') FROM dual;


-- 10.LTRIM和RTRIM *
-- LTRIM  删除左边出现的字符串
-- RTRIM  删除右边出现的字符串
SELECT ltrim('***gao qian jing****','*') FROM dual;
SELECT rtrim('***gao qian jing****','*') FROM dual;
SELECT ltrim(rtrim('****gao qian jing****','*'),'*') FROM dual;

-- 11.SUBSTR(string,start,count) *
-- 取子字符串,从start开始,取count个     (学号162054225)
SELECT substr('123456789',3,8)  FROM  dual;

-- 12.REPLACE('string','s1','s2')
--   string   希望被替换的字符或变量
--   s1       被替换的字符串
--   s2       要替换的字符串
SELECT replace('He love you','He','I') FROM dual;

-- 13. TRIM('s' FROM 'string')
--       LEADING   剪掉前面的字符
--       TRAILING  剪掉后面的字符
select trim('a' FROM 'a b 123') FROM dual;

-- 14. ABS
-- 返回指定值的绝对值
SELECT abs(100),abs(-100) FROM dual;

-- 15. CEIL *
-- 返回大于或等于给出数字的最小整数
SELECT ceil(3.1415927) FROM dual;

-- 16. FLOOR *
-- 对给定的数字取整数
SELECT floor(2345.67) FROM dual;
```

```sql
-- 17. MOD(n1,n2) *
--  返回一个n1除以n2的余数
SELECT mod(10,3),mod(3,3),mod(2,3) FROM dual;

-- 18. ROUND *
-- 按照指定的精度进行舍入(对比20)
SELECT round(55.5),round(-55.4),trunc(55.5),trunc(-55.5) FROM dual;
SELECT round(3.1415926,3),round(-3.1415926,3),trunc(3.1415926,3),trunc(-3.1415926,3) FROM dual;

-- 19.SIGN *
-- 取数字n的符号,大于0返回1,小于0返回-1,等于0返回0
SELECT sign(100),sign(-100),sign(0) FROM dual;

-- 20.TRUNC *
-- 按照指定的精度截取一个数
SELECT trunc(124.1666,-2) trunc1,trunc(124.16666,2) FROM dual;

-- 21.ADD_MONTHS *
-- 增加或减去月份
SELECT to_char(add_months(to_date('200712','yyyymm'),2),'yyyymm') FROM dual;

SELECT to_char(add_months(to_date('200712','yyyymm'),-2),'yyyymm') FROM dual;

-- 22. LAST_DAY
-- 返回日期(当月)的最后一天
SELECT last_day(to_date('2007-05-12','yyyy-mm-dd'))FROM dual;
SELECT to_char(to_date('2007-05-12','yyyy-mm-dd'),'yyyy-mm-dd') FROM dual;
SELECT to_char(last_day(to_date('2007-05-12','yyyy-mm-dd')),'yyyy-mm-dd')FROM dual;
SELECT to_char(SYSDATE,'yyyy-mm-dd hh24-mi-ss day') FROM dual;
SELECT to_char(last_day(SYSDATE),'yyyy-mm-dd') FROM dual;

-- 23.NEXT_DAY(date,'day')
-- 给出日期date和星期x之后计算下一个星期的日期
SELECT next_day('08-8月-2007','星期三') next_day FROM dual;

-- 24.SYSDATE  **
-- 用来得到系统的当前日期
SELECT to_char(sysdate,'yyyy-mm-dd day') FROM dual;

-- 25. TO_CHAR(date,'format') **
SELECT to_char(sysdate,'yyyy/mm/dd hh24:mi:ss day') FROM dual;
SELECT to_char(sysdate,'yyyy-mm-dd hh-mi-ss am') FROM dual;

-- 26. TO_DATE(string,'format') **
-- 将字符串转化为ORACLE中的一个日期
SELECT to_date('2003-09-08 18:26:37', 'yyyy-mm-dd hh24:mi:ss') from dual;

-- 27. TO_NUMBER *
-- 将给出的字符转换为数字
SELECT to_number('2019') year FROM dual;

-- 28. GREATEST *
-- 返回一组表达式中的最大值,即比较字符的编码大小.
SELECT greatest('AA','AB','AC') FROM dual;
SELECT greatest('a','b','c') FROM dual;
```

```sql
-- 29. LEAST *
-- 返回一组表达式中的最小值
SELECT least('啊','安','天') FROM dual;
SELECT least('a','b','c') FROM dual;

-- 30.UID
-- 返回标识当前用户的唯一整数
SELECT uid FROM dual;
SELECT uid FROM emp;

-- 31.USER
-- 返回当前用户的名字
SHOW user;
SELECT user FROM  dual;

-- 32.AVG(DISTINCT|ALL) *
-- all表示对所有的值求平均值,distinct只对不同的值求平均值
SELECT avg(distinct sal) FROM emp;
SELECT avg(all sal) FROM emp;

-- 33.MAX(DISTINCT|ALL) *
-- 求最大值,ALL表示对所有的值求最大值,DISTINCT表示对不同的值求最大值,相同的只取一次
SELECT max(distinct sal) FROM emp;

-- 34.MIN(DISTINCT|ALL) *
-- 求最小值,ALL表示对所有的值求最小值,DISTINCT表示对不同的值求最小值,相同的只取一次
SELECT min(all sal) FROM emp;

-- 35.GROUP BY *
-- 主要用来对一组数进行统计
SELECT deptno,count(*),sum(sal) FROM emp group by deptno;

-- 36.HAVING *
-- 对分组统计再加限制条件
SELECT deptno,count(*),sum(sal) FROM emp group by deptno
having count(*)>=5;

-- 37. ORDER BY *
-- 用于对查询到的结果进行排序输出
SELECT deptno,ename,sal FROM emp order by deptno,sal desc;

-- 38. Extract(date FROM datetime)
-- 从日期时间中取得特定数据（例如取出月、年等）
SELECT extract(YEAR FROM sysdate)   FROM dual;
SELECT extract(MONTH FROM sysdate) FROM dual;
SELECT extract(DAY FROM sysdate)    FROM dual;
SELECT extract(HOUR FROM localtimestamp) FROM dual;
SELECT extract(MINUTE FROM localtimestamp) FROM dual;
SELECT extract(SECOND FROM localtimestamp) FROM dual;

--39. DECODE(expr,search1,result1[,search2,result2][,default])**
-- 多值判断，如果某一列的数据与判断值相同，则使用指定的显示结果输出，如果没有满足条件，则显示默认值
SELECT deptno,DECODE(deptno,'10','ABC','20','DEF','30','AAA','NO') result FROM dept;

-- 40.NULLIF(expr1,expr2)
-- 如果expr1和expr2的值相等，则返回NULL，否则返回expr1
```

```
SELECT NULLIF('abc','abc') result FROM dual;
SELECT NULLIF('abc','abb') result FROM dual;


-- 41.NVL(expr1,expr2)**
-- 如果expr1是null,则返回expr2; 如果expr1不为null,则返回expr1
SELECT empno,ename,comm,NVL(comm,0) FROM emp;


-- 42. NVL2(expr1,expr2,expr3)
-- 如果expr1不是null,则返回expr2;如果expr1是null,则返回expr3;参数expr1可以是任意数据类型，而expr2和expr3可以是除Long之外的任何数据类型
SELECT empno,ename,comm,NVL2(comm,1,0) FROM emp;

-- 43.Translate(x,from_string,to_string) *
-- 从'a1b0c2d3'中找在‘3210’中出现过的字符，找到后用‘pqxy’相应位置的字符替换
-- 实际使用中最好使第二个参数 和 第三个参数的位数一样
-- 0 用 y 代替，1 用 x 代替，2 用 q 代替，3 用 p 代替
SELECT translate('a1b0c2d3','3210','pqxy') from dual;
SELECT translate('32100123','3210','pqxy') from dual;
```

# 第三章 Java SE

## 第一节：Java开发起步

### 一、前期准备

#### （一）下载安装JDK

Jdk下载官网

#### （二）配置环境变量

需要配置的两个环境变量：

1. Path:可执行文件路径(路径指向jdk的bin路径)
2. CLASSPATH：类路径，值是 .;

> 分号不能是汉字输入法下的
>
> - 问题：Path和CLASSPATH大小写区分吗？ 在Windows下不区分，在Linux, Unix, Mac下必须是Path,CLASSPATH
> - 问题：Windows环境下，用户变量和系统变量一样，谁会生效？ 用户变量生效，系统变量会被覆盖
> ####（三）验证环境变量配置是否成功

1. 验证方式一：javac, java

- 问题：javac和java从哪里来？ Path环境变量的值，就是JDK的bin目录

2. 验证方式二：set命令(可查可改) 使用set classpath或者 set path，查看显示的是否正确。 在set设置时，设置结果仅对当前控制台或者命令行窗口有效，控制台关闭，set结果失效。

> Win+R 输入sysdm.cpl 进入高级->环境变量 ####（四）编写HelloWorld程序，不带包

- 编写Java代码的工具：Java文件实际上就是文本文件。 eg：记事本、Atom、Notepad++、EditPlus(商业)、Sublime、VS Code
- 注意事项：
    1. 必须加规范的注释(版权注释，JavaDoc注释)
    2. 缩进4个空格
    3. 类名，方法名后加一个空格
    4. Java严格区分大小写
    5. 编译：javac HelloWorld.java

> 有GBK中文问题时，这样输入 javac -encoding utf-8 HelloWorld.java DOS命令： 切换到D盘： D:+enter 显示当前目录： dir 退回到上级目录： cd .. 退回至根目录： cd \ 补全命令 首字母+tab键 ####（五）编写HelloWorld，带包

- 编译方法：javac -d . HelloWorld.java

> "."表示在当前目录下 执行方法：java cn.edu.tit.corejava.Day01.HelloWorld ####（六）在一个Java文件中编写多个类

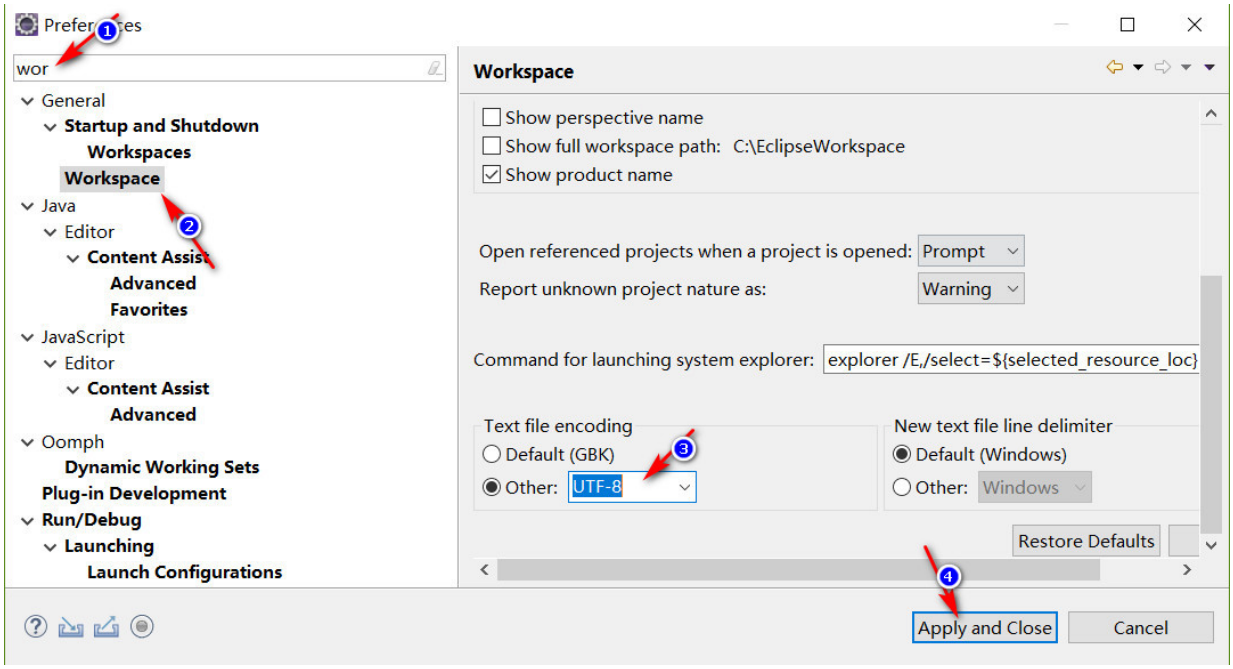
```
public class Person {
        public static void main(String []args) {
                System.out.println("Person");
        }
}

class Student {
        public static void main(String []args) {
                System.out.println("Student");
        }
}

class Teacher {
        public static void main(String []args) {
                System.out.println("Teacher");
        }
}
```

- 问题1：将Person.java另存为Student.java,为什么编译无法通过？ 如果一个类被public修饰时，保存文件时，文件名和此类名命名一致
- 问题2：将Student修饰为public，为什么无法通过编译？ 一个Java文件中，最多只能有一个类被public修饰。 如果所有类都不是public，文件名可以自己命名

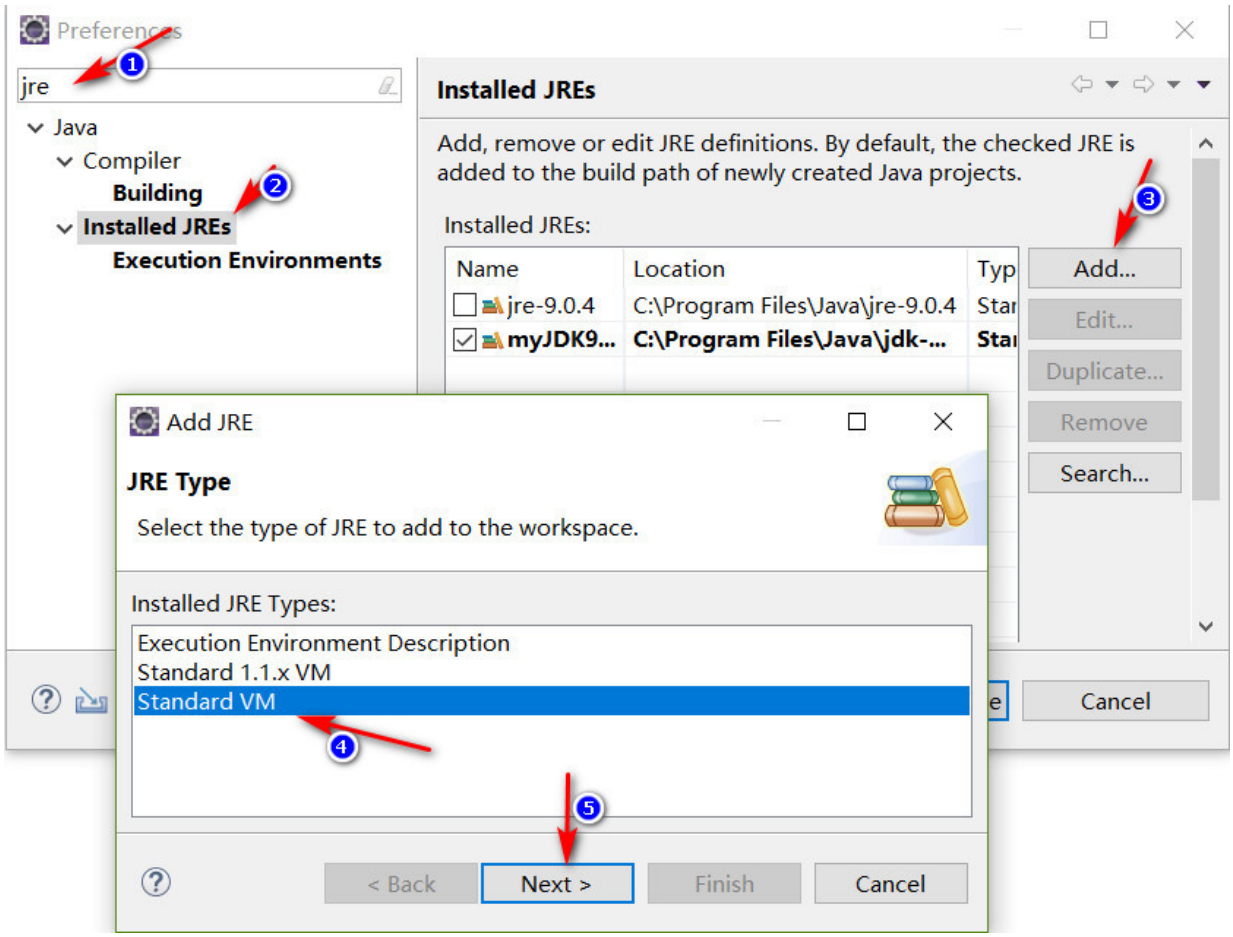
> .class文件叫字节码文件，使用内存地址+16进制编码表示， 以CAFEBABE开头 ####（七）配置并使用Eclipse

- 什么是IDE？集成开发环境
- 主要的IDE有哪些？Eclipse(免费)，IDEA(商业)，NetBeans(免费的，来自sun公司) Eclipse下载官网 从官网上下载后得到一个压缩包，解压即可使用(绿色软件)
- Eclipse使用先决条件： 先安装JDK，并配置好环境变量， 注意：JDK的位数与Eclipse的位数一致 Eclipse的版本和它匹配的JDK版本要一致，如果Eclipse要求JDK8，那么最好是安装JDK8(软件一般向下兼容)
- 术语介绍：
    1. 工作空间 Workspace,存放写好的项目
    2. 工程或者项目，在写程序前必须先建立工程
- 正式使用前必须配置Eclipse

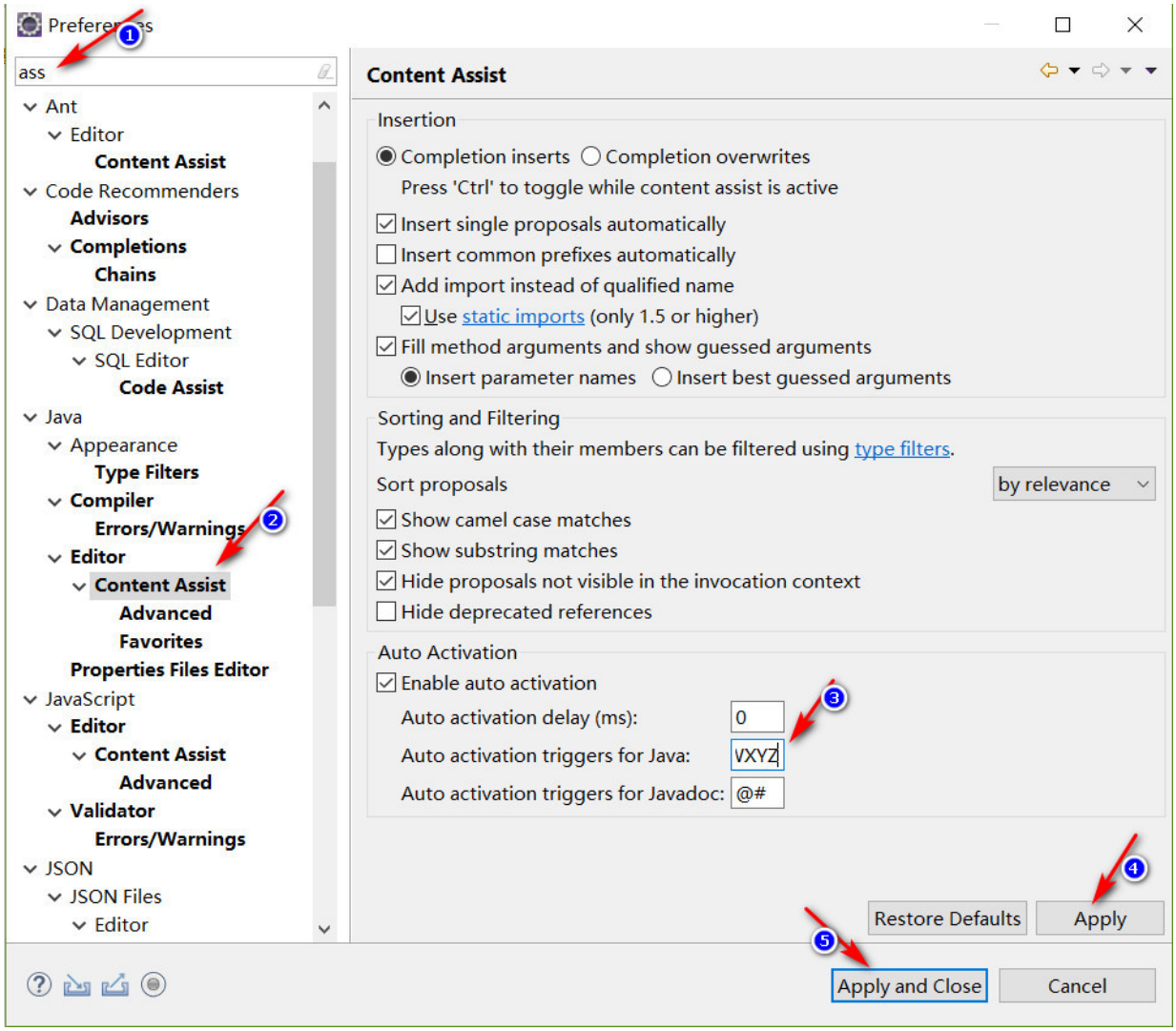
1. 配置编码，utf-8（处理乱码）




2. 配置JDK



3. 配置自动提示




- Eclipse如何使用?
  1. 如何建立工程或者项目?
  2. 如何编写HelloWorld程序? 执行HelloWorld方式：鼠标点击 ctrl + F11 或者 ALT + SHIFT + X,J
  3. Eclipse常用的快捷键

     > 任务：搜索Eclipse快捷键 Ctrl + M: 放大或者还原代码编辑窗口 Ctrl + W: 关闭当前编辑的窗口 Ctrl + Shift + W: 关闭所有编辑窗口 Ctrl + E: 结合上下键，切换编辑窗口 Ctrl + D: 删除光标所在行 Shift + Enter:在光标所在行后，另起一行 Ctrl + Shift + Enter:在光标所在行前，另起一行 Alt + 上下键： 移动代码行的位置 Alt + 左键： 回到前一编辑页面 Alt + 右键： 回到后移编辑页面 Ctrl + /:单行注释 Ctrl + Shift + /:多行注释 Ctrl + Shift + \:取消多行注释 Ctrl + L: 快速定位行 Ctrl + Q: 回到上次操作行 Ctrl + Shift + F:格式化代码(代码无error,搜狗输入法关闭) Alt + Shift + S: 自动补充代码 Ctrl + Alt + 下键： 复制当前行到下一行 Ctrl + Alt + 上键： 复制当前行到上一行 Ctrl + Shift + P:定位到下一匹配符(括号的匹配)

  4. 如何安装Eclipse插件(plugin)?
     - 打开文件路径的插件 OpenExplorer 复制插件的jar包，放置到Eclipse的plugins目录中； 重启Eclipse，如果不生效，删除Eclipse目录下的configuration的org.eclipse.update，重启若还是不生效可能是版本不匹配
     - 安装阿里巴巴的代码审查工具 Help->Install New Software-> 将"https://p3c.alibaba.com/plugin/eclipse/update"这段网址复制在Work with目录下 ->点击Add(为工具命名)->next->select all勾选上->一路next安装完毕 使用： 右键->选择"阿里编码规范检查"->如有不规范代码会显示出来

# 二、了解Java

## （一）HelloWorld程序解析

args 是arguments的缩写，一般默认写为args，是可以修改的 String[] args 这个字符串数组是保存运行main函数时输入的参数的

1. 去掉类前面的修饰符，程序能否编译和执行?
    - 去掉类前public：编译通过，执行通过!
2. 去掉main前面的修饰符，程序能否编译和执行?
    - 去public：能通过编译，不能执行(找不到main方法)
    - 去static: 能通过编译，不能执行(main 方法不是类 HelloWorld 中的static)
    - 去void: 不能通过编译(方法声明无效，需要返回值类型)
3. main前面的修饰符调换位置，程序能否编译和执行?
    - public与static: 编译通过，执行通过!
    - void与main: 不能通过编译
    - static与void：不能通过编译
    - public与void：不能通过编译
4. 去掉main中形参，程序能否编译和执行?
    - 空：能通过编译，不能执行(找不到main方法)
    - 去String: 不能通过编译(非法类型开始)，不能执行(找不到main方法)
    - 去args: 不能通过编译(需要标识符)，不能执行(找不到main方法)
    - 去[]：能通过编译，不能执行(找不到main方法)
5. 将main形参中的方括号换成...（三个小数点），程序能否编译和执行?
    - 可以通过编译，能正常执行

## （二）查看Java发展历史

附上一个链接，写的很全面 Java发展历史
Java官方文档

## （三）JDK、JRE和JVM的关系

1. JDK(Java Development Kit): 是Java程序开发工具包，包含JRE和开发者用的工具。 可以运行也可以开发Java程序
2. JRE(Java Runtime Enviroment): 是Java程序的运行环境，包含JVM标准实现和运行时所需的核心类库。 可以运行Java程序
3. JVM(Java Virtual Machine): Java虚拟机，是运行所有Java程序的假想计算机，是Java程序的运行环境。 Java的虚拟机本身不具有跨平台的性能，每个操作系统下有专属的JVM。 所有编写的Java程序都运行在JVM上,可将.class文件解释为机器码（...\jdk\jre\bin\java.exe）

> JDK中包含JRE，在JDK的安装目录下有一个名为jre的目录，里面有两个文件夹bin和lib，在这里可以认为bin里的就是jvm，lib中则是jvm工作所需要的类库，而jvm和 lib和起来就称为jre。
>
> ```
> 疑问：JDK目录下有bin、lib、jre文件夹，而其中的jre文件夹下又有bin、lib----有什么不同?
> 答:
>   1. jdk是开发时环境，jre是运行时环境。就是编写java文件用jdk，运行java文件用jre;
>   2. jdk的bin下有各种java程序需要用到的命令，与jre的bin目录最明显的区别就是jdk下才有javac
> ```

## （四）Java程序的执行过程

1. 编写Java文件，得到.java文件(文本文件)

2. 编译器编译，得到.class文件(字节码文件)
3. 解释器将字节码文件编译为特定平台下的机器码
4. 操作系统运行机器码得到运行结果

> javac编译`.java文件`生成`.class文件`，Java翻译成对应操作系统(OS)机器码，操作系统执行机器码

### （五）Java平台的划分

- 参考博客

1. JavaSE Java Standard Edition Java 标准版
2. JavaME Java Micro Edition Java 微型版
3. JavaEE Java Enterprise Edition Java 企业版

### （六）Java语言有哪些特点?

1. 面向对象，支持继承，封装和多态；除简单的变量类型外，绝大多部分成员都是对象。
2. 可移植，一次编译可在不同平台上解释执行
3. 安全性，无指针，有类装载器，字节码校验器，安全管理器等保证Java的安全
4. 并发性，支持多线程并行机制，分布式应用程序

### （七）C/C++/Java的区别?

- Java语言特性

1. 语言：C纯面向过程语言；C++可面向过程可面向对象，是一门过渡语言；Java是面向对象的一门语言；
2. 运行：C/C++ 编译成本地机器码, 可以由操作系统直接运行，效率高；Java编译为字节码，由JVM去解释运行，效率低；
3. 指针：C/C++保留指针，程序员可以直接获取使用内存地址，Java出于安全性考虑，去掉指针;
4. 垃圾回收：C/C++程序员自己回收，手动管理；Java有自动回收机制;
5. 用途：C/C++效率高，适合做底层开发，如操作系统；Java安全性好跨平台,适合做大型软件，APP；

# 第二节：如何设计和使用类

## 一、设计类

- [x] 任务1：设计教务系统中的学生类

> UML：(Unified Modeling Language)统一建模语言
>
> - 使用UML工具：SoftwareIdeasModeler
> - 包含属性和方法
> - 先设计，在开发

1. 类（类型缩写） class

- 类名
- 属性（Attribute）
- 方法 (Operation) ---函数、操作

2. 对象（类的一个实体：实例化一个对象）---万物皆对象

## 二、类的创建与使用

1. 对象如何创建？ new 构造方法

```
Person person = new Person();
```

2. OOP编程主线 类---创建对象---对象.方法、属性
3. 什么构造方法

- 名字和类名相同
- 无返回值类型

```
class Person {
    public void Person() {
      ....
    }
}
// 以上不是构造方法，构造方法没有返回类型（Void 也算是返回类型）
```

4. 默认构造方法

- 我们在创建一个类的时候，可以不明确提供构造方法，JVM会给自动添加构造方法
- 如果提供了构造方法，则JVM不会再自动创建构造方法

```
// 默认构造方法
Person() {

}
```

5. 构造方法的作用是什么?

- 与new配合，创建对象---分配内存空间
- 初始化对象属性默认值

6. 能不能用对象调用构造方法？（不能）
7. private修饰的属性，只能在本类中

- [x] 任务3 如何在别的类中使用Student类的访问类型为private的属性？（封装）

1. 封装的目的：保护数据
2. 如何实现：属性私有，提供公共方法使用属性

属性的默认值问题

- 对象默认：null
- 整数默认：0
- 浮点型默认：0.0
- 字符型默认0
- 布尔型默认false

```
String name = "tit";//显示赋值
int age;//默认赋值
/*
鄙视题：属性值的变化过程
1. 默认赋值
2. 显示赋值
3. 构造方法赋值
4. set()方法
*/
String name = "tit";
new Student("nuc");
//name值变化过程：null---tit---nuc
```

- [x] 课内任务：写一个类，Teacher,提供属性age和name, 对age和name赋值，并输出他们的值，其中age的范围是20~150； 要求：有规范的JavaDoc注释，并生成。

# 第三节：数据类型和变量

【参考：Oracle-java官网】

## 一、数据类型有哪些？ （11种）

- 基本数据类型/（简单数据类型） 8种
- 引用数据类型/（复合数据类型） 3种

### （一）基本数据类型（8种）

1. byte（1字节，8位，-128~127）

```
byte age = 200;//error：超出范围
```

2. short（2字节，16位，-32768~32767）
3. int（4字节，32位，-2^31 ~ 2^31-1）
4. long（8字节，64位，-2^63 ~ 2^63-1）

```
//笔试题：long类型数L、l可有可无（错）
long salary = 2147483647;
long salary = 2147483648;//error
long salary = 2147483648L;//超过int表示的范围后必须要加"L"

/*
注：
  当值小于等于int类型最大值，可不加L,否则必须
  不建议使用小写"l"
*/
```

5. char (2字节，16位，0~65565)

```
char sex = '男';//right
char sex = "男";//error
char sex = 97;//right
char sex = '\u0097'//right:Unicode编码

/*
注:
  字符型数据本质就是数字
  字符型用单引号
  Java语言采用Unicode编码，一个字符就是两个字节
*/
```

- [ ] 不同类型编码：ASCII ,ISO8859-1,GB2312,GBK,Unicode,UTF-8（*详细见Task/README.md*） 链接：https://blog.csdn.net/Fly_1213/article/details/85249201
- [ ] 转义字符：转义就是失去特殊含义 （ 所有的ASCII码都可用"\"加数字（一般是8进制数字）来表示。而C语言中定义了一些字母前加"\"来表示常见的那些不能显示的ASCII字符如\0,\t,\n等，就称为转义字符，因为后面的字符，都不是它本来的ASCII字符意思了。）

|转义字符|意义|ASCII码值（十进制） | :--:| :--:|:--:| |\n| 换行(LF)，将当前位置移到下一行开头|010 |\t| 水平制表(HT)（跳到下一个TAB位置）| 009 |\r| 回车(CR)，将当前位置移到本行开头| 013 |\ \| 代表一个反斜线字符'\' |092 |\"|代表一个双引号字符|034 |\?| 代表一个问号| 063 |\0|空字符(NUL)|000 |\a|响铃(BEL)|007 |\b|退格(BS)，将当前位置移到前一列|008 |\f|换页(FF)，将当前位置移到下页开头|012

```
//打印双引号
System.out.println("\"\"");
//打印两个反斜杠"\\"
```

6. float（4字节，32位，……）

```
flout height = 1.75;//error:必须加f/F
flout height = 1.75F;//right
float height = .8;//right:0.8
float height = 8.;//right:8.0
/*
  注:
    float类型数据"F"/"f"不能省去
*/
```

7. double (8字节，64位，……)
8. boolean 取值：true和false。长度不明确

```
boolean a = 1;//error:只能是True或False
```

**小结:**

- 整数默认int,小数默认double
- Java长度由小变大

```
byte---short---int---long---float---double
//长度不会随平台变化
```

- 数据类型可以转换（从低级类型到高级类型）

```
/*
  规则：低级给高级自动转换，高级给低级强制转换
        排序：byte---short---int---long---float---double
        补充：char可直接转换int,boolean不参加转换
*/
  int num1 = 1;
  double num2 = 2;
  //从低级类型到高级类型
  num2 = num1;//right：自动转换
  //从高级类型转到低级类型
  num1 = num2;//error:需要强制转换
  uum1 = (int)num2;//right
  /*
  注：
    char与short都是16bit，在Java中，char是16bit长的无符号整数，因此范围是：0～65535，而short则是16bit长的带符号整数，范围是-32768～32767。 所以，两者虽然字长相同，但意义却不一样（表示范围并不完全重合或包含），因此不管是char转换成short，还是short转换成char，都属于数值的“窄化转换”（Narrowing Primitive Conversion）。 窄化转化都可能“丢失精度”，因此必须强制转换。
  */
```

### （二）复合数据类型（引用数据类型）

- 复合(引用)类型（3种）： 类、数组、接口
- 引用类型就是类或接口类型 `String name; Address addr;`
- 每个类就是一种类型，因此java中的类型可以引申为无限多种类型 `Person p;`
- java中每个变量都有确定的类型 `int age; Student s;`

## 二、Java值传递

- 方法调用时，实参传给形参
- 传递的参数可以是：基本类型、引用类型
- Java中只有值传递
- 注意堆区和栈区的数据变化（画图）
- 可通过DeBug调试查看变量值

> Eclipse之DeBug调试
>
> 1. 设置断点
> 2. 进入子方法（Step Into/F5）
> 3. 单步执行（Step Over/F6）
> 4. 从子方法跳出（Step Return F7）---结合F5使用
> 5. 结束调试---点击红色按钮
> 6. 回到编辑模式---点击eclipse右上角图标（java）

## 三、面试问题

1. 什么是标识符？命名规则是什么？

- 标识符(identifier)就是名字
- 以"_"、"$”、字母开头，后面加字母、数字、下划线、$

```
//以下哪些是合法的标识符？BCDE
A. Hello World   B.True   C.String   D.学生  E._  F.*P  G.12abc  J.A@B
```

2. 什么是关键字（保留字）？有什么特点?

- 关键字-官网
- 被Java语言保留，有特殊含义，标识符不能用
- 关键字全小写
- true、false、null 不是关键字（参考官网）
- goto、const是关键字，但不能使用
- system不是关键字

```
//面试题：  以下哪些是Java语言关键字？ab de(官方文档不在关键字列表中)
A. goto  B. const C. NULL D. true E. false F. System
```

3. 局部变量和成员变量的区别

- 成员变量就是属性
- 局部变量在方法内部定义
- 局部变量没有默认值，使用前必须先赋值
- 区别方法：看定义的位置
- 开发中成员变量和局部变量名字不要相同

4. 内存逻辑的划分

- 栈：先进后出,存放所有局部变量，由操作系统管理，用于数据交换
- 堆：存放new关键字创建的对象
- 全局数据区：存放全局变量
- 方法区：存放程序代码

5. 什么是Java中的垃圾回收机制？（*详细见Task/README.md*）

- 图解Java 垃圾回收机制
- 垃圾：废弃/无用的对象
- Java中采取自动的垃圾回收机制(garbage collection，GC)---有延时
- 内存泄漏：指废弃的对象没有被及时的回收
- 垃圾回收机制是JVM自带的一个线程（伴随主程序运行着的程序），用于回收废弃的对象
- 目的：减轻程序员的负担，减少了内存泄漏
- 我们能不能通过调用System.gc()强制回收？（不能，仅仅是建议JVM马上调用GC进行垃圾回收）

- [ ] finalize()方法有什么用？----【tip:搜索】（*详细见Task/README.md*）

> 内存泄漏：（Memory Leak）是指程序中己动态分配的堆内存由 于某种原因程序未释放或无法释放，造成系统内存的浪费，导致程 序运行速度减慢甚至系统崩溃等严重后果。

6. Java中是否有可能存在代码泄露?

- 不可避免
- 当长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是java中内存泄露的发生场景。

7. String 是不是内置的数据类型?

- 不是，是Java API中的一个类

# 四、补充

## （一）不同类型编码：ASCII ,ISO8859-1,GB2312,GBK,Unicode,UTF-8

参考连接

- **ASCII**：计算机最初是由美国等国家发明的，所以最初的字符编码规范是以英文为基础。录入英文字符编码标准：ASCII(American Stardard Code for Information InterChange 美国信息互换标准代码)，单字节编码，使用单字节8位 前127个存储英文字母及标点符号。
- **ISO8859-1 / Latin-1**： ISO(国际标准化组织)在ASCII基础上做的扩展，向下兼容ASCII，也是单字节编码，是许多欧洲国家使用的编码标准。
- **GB2312**： 随着发展，各国都需要使用各国的语言使用计算机，相应的，我国大陆地区采用双字节 高低位字节的方式，存储简体中文。当存储中文字符时，高低位都存放大于127的数字，当读取高位字节时，若大于127，则认为是中文字符，将读取双字节进行识别，若小于127，则认为是英文字符，单字节识别。另外，GB2312在大于127位的编码库也提供了中文符号的编码，故英文标点符号称做半角符号，中文标点符号称作全角符号。
- **GBK**： 对GB2312的拓展，双字节编码，区别是当存储中文字符时，高位存放大于127的数字，低位不再是大于127的数字，将127之前的数字也囊括了。GBK将繁体中文也囊括进来。
- **Unicode**： 针对各个国家都使用不同的编码机制，iso(国际标准化组织)又站了出来，对全球编码做了整个的囊括，推出了Unicode ，但其字符集也是最复杂、占用空间最大的。 Unicode，定长双字节编码，对ASCII采取高位补零的方式拓展，不兼容iso8859-1编码。 说明： 定长编码便于计算机处理（GB2312/GBK不是定长编码），而unicode又可以用来表示所有字符，所以在很多软件内部是使用unicode编码来处理的，比如java。
- **UTF-8**： 由于Unicode是一组编码映射，即一个字符映射一个16进制数字的形式。随着互联网发展，unicode不便于传输和存储，为了节省资源（无论是硬盘存储还是流量），相应的就产生了utf-8编码。 UTF-8兼容ISO8859-1编码，同时也可以用来表示所有语言，不过UTF-8编码是不定长编码，一般来讲，英文字母都是用一个字节表示，而汉字使用三个字节，比 unicode编码节省空间。

## （二）转义字符

|转义字符|意义|ASCII码值（十进制） | :--:| :--:|:--:| |\n| 换行(LF) ，将当前位置移到下一行开头|010 |\t| 水平制表(HT)（跳到下一个TAB位置）| 009 |\r| 回车(CR) ，将当前位置移到本行开头| 013 |\ \| 代表一个反斜线字符'\' |092 |\"|代表一个双引号字符|034 |\?| 代表一个问号| 063 |\0|空字符(NUL)|000 |\a|响铃(BEL)|007 |\b|退格(BS) ，将当前位置移到前一列|008 |\f|换页(FF)，将当前位置移到下页开头|012

## （三）什么是Java中的垃圾回收机制?

- 参考连接:垃圾回收机制
- 参考连接:图解Java 垃圾回收机制

1. 垃圾回收机制（有延时） 1）立即回收器(Garbage Collection ,GC)是JVM自带的一个线程（伴随主程序运行着的程序），用于回收废弃的对象; 2）Java程序员不用亲自回收废弃对象，因为垃圾回收机制会自动进行回收;
2. 触发GC的条件 1）GC在优先级最低的线程中运行，一般在应用程序空闲即没有应用线程在运行时被调用(存在延时)。但下面的条件例外。 2）Java堆内存不足时，GC会被调用。当应用线程在运行，并在运行过程中创建新对象，若这时内存空间不足，JVM就会强制调用GC线程。若GC一次之后仍不能满足内存分配，JVM会再进行两次GC，若仍无法满足要求，则JVM将报“out of memory”---内存泄漏的错误，Java应用将停止。
3. 两个重要方法 1）System.gc()方法 使用System.gc()可以不管JVM使用的是哪一种垃圾回收的算法，都可以请求Java的垃圾回收。在命令行中有一个参数-verbosegc可以查看Java使用的堆内存的情况，它的格式如下：java -verbosegc classfile 由于这种方法会影响系统性能，不推荐使用，所以不详诉。 2） finalize()方法 在JVM垃圾回收器收集一个对象之前，一般要求程序调用适当的方法释放资源，但在没有明确释放资源的情况下，Java提供了缺省机制来终止该对象心释放资源，这个方法就是finalize （）。它的原型为：protected void finalize() throws Throwable 在finalize()方法返回之后，对象消失，垃圾收集开始执行。原型中的throws Throwable表示它可以抛出任何类型的异常。

> 之所以要使用finalize()，是存在着垃圾回收器不能处理的特殊情况。例如： 1）由于在分配内存的时候可能采用了类似 C语言的做法，而非JAVA的通常new做法。这种情况主要发生在native method中，比如native method调用了C/C++方法malloc()函数系列来分配存储空间，但是除非调用free()函数，否则这些内存空间将不会得到释放，那么这个时候就可能造成内存泄漏。但是由于free()方法是在C/C++中的函数，所以finalize()中可以用本地方法来调用它。

> 以释放这些"特殊"的内存空间。 2）又或者打开的文件资源，这些资源不属于垃圾回收器的回收范围。

4. 减少GC开销的措施 1)不要显式调用System.gc()。此函数建议JVM进行主GC,虽然只是建议而非一定,但很多情况下它会触发主GC,从而增加主GC的频率,也即增加了间歇性停顿的次数。大大的影响系统性能。
   2)尽量减少临时对象的使用。临时对象在跳出函数调用后,会成为垃圾,少用临时变量就相当于减少了垃圾的产生,从而延长了出现上述第二个触发条件出现的时间,减少了主GC的机会。 3)对象不用时最好显式置为Null。一般而言,为Null的对象都会被作为垃圾处理,所以将不用的对象显式地设为Null,有利于GC收集器判定垃圾,从而提高了GC的效率。 4)尽量使用StringBuffer,而不用String来累加字符串。由于String是固定长的字符串对象,累加String对象时,并非在一个String对象中扩增,而是重新创建新的String对象,如Str5=Str1+Str2+Str3+Str4,这条语句执行过程中会产生多个垃圾对象,因为对次作"+"操作时都必须创建新的String对象,但这些过渡对象对系统来说是没有实际意义的,只会增加更多的垃圾。避免这种情况可以改用StringBuffer来累加字符串,因StringBuffer是可变长的,它在原有基础上进行扩增,不会产生中间对象。
   5)能用基本类型如Int,Long,就不用Integer,Long对象。基本类型变量占用的内存资源比相应对象占用的少得多,如果没有必要,最好使用基本变量。 6)尽量少用静态对象变量。静态变量属于全局变量,不会被GC回收,它们会一直占用内存。
   7)分散对象创建或删除的时间。集中在短时间内大量创建新对象,特别是大对象,会导致突然需要大量内存,JVM在面临这种情况时,只能进行主GC,以回收内存或整合内存碎片,从而增加主GC的频率。集中删除对象,道理也是一样的。它使得突然出现了大量的垃圾对象,空闲空间必然减少,从而大大增加了下一次创建新对象时强制主GC的机会。
5. Java程序的内存泄漏问题（不可避免） 1）内存泄漏是指废弃的对象没有被及时的回收; 2）严重的内存泄漏会导致内存中的废弃对象越来越多，直到内存占满程序崩溃; 3）垃圾回收机制判断对象何时回收的依据是该对象是否还有变量在引用; 4）建议：确定一个引用变量的对象不在使用时，应该及时将引用类型变量设置为null;

### （四）java finalize方法总结，GC执行finalize的过程

- 参考：java finalize方法总结、GC执行finalize的过程

1. finalize()方法定义在Object类中，所以所有类都继承了finalize()
2. 当应用程序或 Applet 退出时，调用每个对象的finalize() 方法
3. 调用时间：垃圾回收器在确定某对象没有被引用时(删除之前)，就对此对象调用方法
4. 主要用途：回收特殊渠道申请的内存，如非Java程序(C/C++)申请的内存
5. 不建议使用finalize(),如果用：尽量简单，且要避免对象再生

# 第四节：语句与运算符

## 一、运算符类型

- 算数运算符

```
+   -   *   /   %     ++    --
```

- 比较（关系）运算符

```
   >   >=  <   <=  ==  !=
```

- 逻辑运算符

```
   &&逻辑与(短路与)     ||逻辑或（短路或）      ！逻辑非
   &与                  |或
```

- 位运算符

```
 >>     <<     >>>      ~      ^       &     |
 3 >> 2  //00000011     把左边二进制数字向右移动两位，高位补 符号位
 3 << 2  //00001100     把左边二进制数字向左移动两位，  低位一律补零
 1011 0100 >>> 3        高位一律补零，无符号位右移
面试题$$$:
short x = 10;
x = x + 1;  //x，1是int型，x是short型，运算就高不就低，计算得到int型，而等号左边为short型

short x = 10;
x += 1;      //√，不会报错，可以理解为没有经过内存的转化，直接完成赋值操作

&        同时为真才为真
|        一者为真就为真
^        相同为假不同为真      （相同数字做^运算结果为0）
~        按位取反
<<       左移运算符，低位一律补零
>>       右移运算符，高位补符号位
>>>      右移补零运算符，高位一律补零
```

- 赋值运算符

```
=    扩展的赋值运算符：+=   -=   *=
面试题：
short x =9;
x = x + 1; //error 1为int型,开辟新的内存,存在类型转换

short x = 9;
x += 100; //理解为：x = x + 100;但没有开辟新内存
```

# 二、包

> API(application program interface) :就是类库

1. 使用包的目的：多人开发，不同人不同包，防止类名冲突
2. 报名命名规则：域名倒写+系统名+模块名+层次名。（包名全小写）

```
cn.edu.tit.jobtrace.student.vo
cn.edu.tit.jobtrace.student.dao
cn.edu.tit.jobtrace.student.service
```

3. 导入包

```
import java.util.Scanner;//导入指定类
import java.util.*;//导入所有的类（不推荐）
```

> java.lang 包不需要导入（自动导入的），其余包都要导;

常用的包

- java.io---java输入输出包（文件读写）
- java.lang---java语言的核心包
- java.lang.reflect---反射相关包
- java.util---java实用包

- java.math---
- java.sql---
- java.text

4. 分类

- java基础包:以java开头
- java扩展包：以javax开头
- 第三方包：以org、com等开头
- 自定义包

5. 导包次序：java基础包->java扩展包->第三方包->自定义包

## 三、面试问题

1. if语句后面如果是定义语句，必须加花括号

```
//right
if (5 > 0)
  System.out.println("hello");

//error
if (5 > 0)
  int a = 5;

//right
if (5 > 0) {
  int a = 5;
}
```

2. 结束整个循环有几种方式

```
  break;//只能跳出循环
  return;//程序不在向下执行
  system.exit();//直接退出程序
```

3. 双重循环效率问题

```
for (size_t i = 0; i < 5; i++)
{
  for (int j = 0; j < 3; j++)
  {

  }
}
//比上面效率高
for (size_t i = 0; i < 3; i++)
{
  for (int j = 0; j < 5; j++)
  {

  }
}
```

4. 扩展的赋值运算符号特点

```
//byte、char、shorrt类型在运算的时候自动变成int或者说JVM把他们以int考虑，只要不超范围就可以;
short s = 20;//reght
byte x = 100;//right
s = s + 10;//error右侧是高级类型，赋值给低级类型需要强制转换
s += 10;//right,
```

5. 1-0.9的结果?

- 不是0.1，接近0.1
- 十进制和二进制转化在浮点数运算上存在精度丢失

```
System.out.println("1-0.9 = "+(1-0.9));
System.out.println("1-0.8 = "+(1-0.8));
System.out.println("1-0.7 = "+(1-0.7));
System.out.println("1-0.6 = "+(1-0.6));
System.out.println("1-0.5 = "+(1-0.5));
System.out.println("1-0.4 = "+(1-0.4));
System.out.println("1-0.3 = "+(1-0.3));
System.out.println("1-0.2 = "+(1-0.2));
System.out.println("1-0.1 = "+(1-0.1));
```

6. switch语句

```
switch(n)
{
case 1:
  执行代码块 1
  break;
case 2:
  执行代码块 2
  break;
default:
  n 与 case 1 和 case 2 不同时执行的代码
}
```

- switch语句中可以检测的类型：int、short、char、String、enum
- 不能检测的数据类型为：long、boolean、浮点型（double、float）
- default可有可无，且位置在哪不受影响。

# 第五节：面向对象特性

1. 封装
2. 继承
3. 多态
4. 抽象

> 封装的目的：保护数据 继承的目的：类的复用 多态的目的：增强程序的复用性，解耦。 抽象的目的：被子类继承，实现类的复用

## 一、访问控制

> 访问控制：控制类外面可以访问类中的那些属性和方法；

1. 访问控制符号：用于修饰属性和方法/（类）

   |修饰符|本类内部|同包中的类|子类|包外其它类 |:--:|:--:|:--:|:--:|:--:| |public|可以访问|可以访问|可以访问|可以访问 |protected|可以访问|可以访问|可以访问|不能访问 |默认|可以访问| 可以访问|不能访问|不能访问 |private|可以访问|不能访问|不能访问|不能访问

2. 访问控制符修饰类：

- 对于类的修饰可以使用public和默认（default）方式；
- public修饰的类可以被任何一个类使用；
- 默认的访问控制的类只能被同包中的类使用；

# 二、封装

1. 什么是封装：就是将数据和对数据的操作集中的定义在对象中，外界仅能通过对象提供的接口访问对象的属性和功能;
   - 属性私有(private)
   - 提供公共方法访问(public)
2. 封装的意义：对外提供可调用的、稳定的功能；
3. 封装内部具体的实现细节，外界不可访问，这样的好处在于：降低出错的可能性；
4. 内部变化，不会影响外部使用；

# 三、继承

1. 什么是继承：类的复用---即使用现有类的定义复制并扩展出另一个新的类定义，子类可以继承父类中的成员变量和成员方法(不包含构造)，同时也可以定义自己的成员变量和成员方法；

   > 泛化：从多个类中，抽取相同部分，生成父类的过程叫做泛化； 设计时，从子类泛化出公共父类，再让子类继承父类；

2. Java语言不支持多重继承，一个类只能继承一个父类，但一个父类可以有多个子类；

- 面试题：一个类只能有一个父类？不一定---A->B->C

3. 继承的语法规则？extends

```
Class Person {
  //...
}
Class Student extends Person {
  //...
}
```

4. 继承要求：取决于父类的访问控制符

- 同一包中继承：子类可以访问父类访问控制符为：public、protected、以及默认（default）的属性即方法；
- 跨包继承：默认情况下，只要不在一个包内，即使是继承也无法访问父类的默认访问控制的属性及其方法
  - 类中访问： 若子类需要访问父类属性或方法：要求父类属性及方法访问权限最低要是protected（public、protected）；
  - 类外访问： 子类对象只能直接访问子类从父类继承的访问控制符为【public】的属性或方法； 若子类需要访问父类中protected修饰的方法：子类需要使用重写通过super来调用父类的方法； (java中规定：子类重写父类的方法，子类的方法访问控制范围不能小于父类方法的访问控制范围)

5. 继承关系下，父类的构造方法调用问题?

- 创建子类对象的时候，父类的默认构造方法会默认调用
- 父类没有构造方法时，子类的构造函数需要使用super()指定父类的构造方法
- super()必须放在子类构造方法的第一行

6. 对象的构造次序如何?

- 继承关系是自下而上---子类继承于父类
- 构造函数调用是自上而下---先完成父类的构造，再完成子类的构造

7. 方法的重写（Override）：

- 原因：继承于父类的方法无法满足子类的需要
- 方法名、参数列表（数据类型、个数、次序）、返回值类型相同，访问控制相同或者更加宽泛
- 父类构造方法不能被继承，所以不能被重写

> @Override : 断言机制。就是告诉编译器当前方法是重写父类方法， 请编译器协助检查方法签名，如果重写的方法签名在父类中没有找到，则编译错误;

8. 方法的重载（Overload）：一个类中有多个同名方法

- 规则：方法名字相同，参数列表必须不同，返回值和访问控制符不限
- 构造方法可以重载

9. this的使用:

- this表示当前类的对象
- 使用this()调用本类自己的构造方法
- this.(属性/方法),调用本类的属性和一般方法

10. shper的使用:

- super表示父类的对象
- 使用super()调用父类的构造方法
- 使用super.(属性/方法)，在子类中调用父类的属性和一般方法

> this()和supr()不能同时调用，都需要放在构造方法体的首行

12. 创建子类对象时，程序执行过程?

- 父子类当中只有无参构造方法：先执行父类构造后执行子类构造
- 父类子类中属性赋值动作，赋值也是先赋值完父类的属性值，再赋值子类的属性， 代码执行时先给属性赋值，再执行方法。
- 父子类有游离块：看代码位置

> 属性赋值顺序：隐式初值-->显示赋值-->构造赋值

13. **父类的引用指向子类的对象**

- 父类的引用可以指向子类的对象，但通过父类的引用只能访问父类自己定义的属性和功能部分（包含子类重写父类的方法），不能访问子类扩展的部分（独有的属性和功能）
- 例如Person类型的对象，就无法访问子类Student的成绩这个属性

14. 动态方法调度

- 在运行时，父类变量根据指向子类对象的不同，动态判断调用何种重写方法

```
/*动态方法调度*/
Person dad = null;
Child1 = son1 = new Child1();
Child2 = son2 = new Child2();

//爸爸和大儿子一起做蛋炒饭
dad = son1;
dad.cook;

//爸爸和二儿子一起包饺子
dad = son2;
dad.cook;
```

15. 笔试题（重写和重载的区别&&父类的引用指向子类的对象）

```
//下面代码输出结果是?
class Super {
    public void f() {
        System.out.println("super.f()");
    }
}

class Sub extends Super {
    public void f() {
        System.out.println("sub.f()");
    }
}

class Goo {
    public void g(Super obj) {
        System.out.println("g(Super)");
        obj.f();
    }

    public void g(Sub obj) {
        System.out.println("g(Sub)");
    }
}
class Test{
    public static void main(String[] args) {
        Super obj = new Sub();
        Goo goo = new Goo();
        goo.g(obj);
    }
}

//结果
g(Super)
sub.f()
/*
注意：如果父类Super中没有f()方法，那么这道题的输出结果就是编译错误;
原因是：父类中没有f()方法，子类中的f()方法就不是重写，所以父类定义的引用变量obj就不能访问子类的成员方法/（非重写方法）！
*/
```

# 四、抽象

1. 抽象类、抽象方法

- 抽象类就是仅定义所有子类共享的形式，而没有定义具体实现细节的父类
- 抽象类中，那些只有方法的定义，没有实现细节(没有方法体)的方法叫做抽象方法；
- 一个类中如果包含抽象方法，该类应用abstract关键字声明为抽象类；
- 如果一个类继承了抽象类，必须重写其抽象方法（除非该类也声明为抽象类），且不同的子类可以有不同的实现。
- 语法：（被`abstract`修饰）

```
// 抽象类
public abstract class 类名 {
    // class boby

    // 抽象方法
    访问修饰符 abstract 返回值类型 方法名();
}
```

2. 抽象类不能实例化

- 抽象类不可以实例化，因为抽象类是不完整的，要想使用抽象类中的普通方法，只能通过子类对象继承后调用；

```
Father father =  new Father();//编译错误
```

- 另外，即使一个类中没有抽象方法，也可以使用abstract修饰类。作用就是不让该类实例化；
- abstract VS final □ 抽象类：天生就用来被继承！不继承，就没有任何意义。 □ final：专门用来禁止被继承！和抽象类的意义刚好相反！

> 高内聚低耦合:在一个模块内部 23中设计模式**********

# 五、接口

## （一）定义一个接口

1. 接口是一个标准
2. 接口是引用类型之一；（类、数组、接口）
3. 接口的作用是：解耦，降低耦合,便于维护。
4. 接口是一组方法定义的集合，但所有方法没有实现(默认)；

> JDK1.8之后，接口中的方法可以有方法体（被default,static修饰的方法） 链接

5. 接口天生用来被继承，不能被实例化；
6. 语法：

```
public interface 接口名{
    public abstract 返回值类型 方法名();
}
//public abstract 可以省略，编译器会自动补全；
```

## （二）实现一个接口

1. 一个类可以通过implements关键字“实现”接口，称为：实现类；

2. 语法：

```
public class 实现类类名 implements 接口名{
    //class boby
}
```

3. 一个类要实现某个接口，就必须实现该接口中定义的所有抽象方法；
4. 因为接口中的方法都是public修饰的，所以子类实现接口的方法时，也必须显示定义public访问修饰；
5. 实现类可以实现多个接口，该类需要实现所有接口中定义的所有方法；；

## （三）接口的继承

1. 接口间可以存在继承关系，一个接口可以通过extends关键字继承另外一个接口。子接口继承了父接口中定义的所有方法。
2. 接口支持多继承，一个接口可继承多个接口；

## （四）继承父类与实现接口

1. 一个普通的类可以继承一个类的同时再实现若干接口。
2. 语法：(强调先继承（extends）再实现接口（implements）,不能颠倒！）

```
public class 子类名 extends 父类名 implements 接口 {
    // class body
}
```

## （五）动态调用接口方法

1. 接口可以作为一种特殊类型声明一个引用类型的变量。

```
IUnionPay atm;//编译正确
```

2. 一个接口类型的变量可以引用实现了该接口的类的对象。 因为可以动态调度，所以可以使用接口类型引用实现类的变量；

```
IUnionPay atm = newATMCBC();
IUnionPay atm = new ATMABC();
```

3. 通过接口类型变量仅能调用实现该接口的类中重写的方法。

```
atm.drawMoney(3000);//本质，父类型的引用，可以调用子类型重载的方法；
atm.takePic();//编译错误：因为是子类重写别的接口的方法,不是重写当前IUnionPay接口的，所以不能调用。
```

## （六）接口中定义常量

1. 接口中除了抽象方法外，只能定义“常量”。
2. 编译器会自动增加public static final 修饰。
3. 何时在接口中定义常量：一般用于实现类中引用固定的备选项。
4. 比如：银联规定只能支持CBC,ABC两家银行的银行卡，不支持其它银行的银行卡。
5. 在接口中定义常量，必须声明的同时初始化。

## （七）接口和抽象类的异同点：

1. 相同点

- 天生都是被继承的。不能实例化对象
- 一个类无论是实现接口还是继承抽象类，都必须实现其中所有的抽象方法。
- 接口和抽象类都可以声明父类型的变量，引用子类型对象
- 父类型对象的引用都可以调用子类型对象中重写的方法

> 切记：父类型对象的引用都不能调用子类型对象中非重写的方法

- 因为以上两点，所以抽象类和接口，都可以实现动态创建对象，和动态方法调度

2.不同点

- 抽象类不允许多继承，而接口可以多继承
- 使用场景不同： 抽象类：主要用于封装子类中共享的成员。只有个别方法自己不能实现，才能定义抽象方法，请子类自力更生自己实现 接口：主要用于在程序中定义标准。接口完全不实现任何方法
- 成员的修饰词不同 抽象类中的成员，可以使用任何关键字修饰 接口中的成员变量都是public static final 的，方法都是public abstract的

### （八）面试题

1. 抽象类能否实现接口？ 可以。
2. 接口能实现接口吗？ 不能。
3. 一个类一定是先继承父类，在实现接口

## 六、多态

1. 什么是多态：对象的多种形态，一个对象被多种称谓
2. 表现方面：

- 引用多态（继承或接口实现）

```
// 父类的引用可以指向本类的对象;
Animal obj1 = new Animal();
// 父类的引用可以指向子类的对象;
Animal obj2 = new Dog();
```

> 这里我们必须深刻理解引用多态的意义，才能更好记忆这种多态的特性。为什么子类的引用不能用来指向父类的对象呢？通俗的讲：就以上面的例子来说，我们能说“狗是一种动物”，但是不能说“动物是一种狗”，狗和动物是父类和子类的继承关系，它们的从属是不能颠倒的。当父类的引用指向子类的对象时，该对象将只是看成一种特殊的父类（里面有重写的方法和属性），反之，一个子类的引用来指向父类的对象是不可行的！！

- 方法多态 根据上述创建的两个对象：本类对象和子类对象，同样都是父类的引用，当我们指向不同的对象时，它们调用的方法也是多态的。 （1）创建本类对象时，调用的方法为本类方法； （2）创建子类对象时，调用的方法为子类重写的方法或者继承的方法； （3）使用多态的时候要注意：如果我们在子类中编写一个独有的方法（没有继承父类的方法），此时就不能通过父类的引用创建的子类对象来调用该方法！！！ **注意： 继承（包括接口的实现）是多态的基础。**

3. 多态中方法调用和执行

- 编译时，方法调用看变量：父类引用指向不同的子类对象，所调用的方法随子类变化而变化。
- 运行时，方法执行看对象：如果一个子类型的对象，向上造型为父类型的变量时，向上造型为不同父类型变量可用的功能也是不一样的。

4. 为什么使用多态？(增强代码的复用能力)

```
class Person {
    int age;

    public Person(int age) {
        super();
        this.age = age;
    }

    public int getAge() {
        return age;
    }
}

public class Student extends Person {

    public Student(int age) {
        super(age);
    }

    public void learn() {
        System.out.println("好好学习,天天向上!");
    }
}

class Teacher extends Person {

    public Teacher(int age) {
        super(age);
    }

    public void teach() {
        System.out.println("好好教书! ");
    }
}

class CalcBirthYear {
    public int CalcYear(Person person) {
        return 2019 - person.getAge();
    }
}

class Test {
    public static void main(String[] args) {
        Student student = new Student(20);
        Teacher teacher = new Teacher(30);

        CalcBirthYear calcBirthYear = new CalcBirthYear();
        int syear = calcBirthYear.CalcYear(student);
        int tyear = calcBirthYear.CalcYear(teacher);
        System.out.println("学生出生年: " + syear);
        System.out.println("老师出生年: " + tyear);
    }
}
```

# 七、引用类型转换

1. 向上类型转换(隐式/自动类型转换，小类型转换为大类型)

- 父类应用指向子类对象时就是向上类型转换

```
Dog dog = new Dog();
Animal animal = dog;// 自动类型提升，向上类型转换
```

2. 向下类型转换(强制类型转换，大类型转换为小类型)

- 将上述代码再加上一行，我们再次将父类转换为子类引用，那么会出现错误，编译器不允许我们直接这么做，虽然我们知道这个父类引用指向的就是子类对象，但是编译器认为这种转换是存在风险的。如：

```
Dog dog = new Dog();
Animal animal = dog;// 自动类型提升，向上类型转换
Dog dog2 = animal;// 编译报错
```

- 我们可以通过强制类型转换。如：

```
Dog dog = new Dog();
Animal animal = dog;// 自动类型提升，向上类型转换
Dog dog2 = (Dog)animal;// 向下类型转换，强制类型转换
```

- 但是如果父类引用没有指向该子类的对象，则不能向下类型转换，虽然编译器不会报错，但是运行的时候程序会出错，如：

```
Dog dog = (Dog)new Animal();// 编译无错，执行报错!
// 其实这就是上面所说的子类的引用指向父类的对象，而强制转换类型也不能转换！！
```

- 还有一种情况是父类的引用指向其他子类的对象，则不能通过强制转为该子类的对象。如：

```
Dog dog = new Dog();
Animal animal = dog;// 自动类型提升，向上类型转换
Dog dog2 = (Dog)animal;// 向下类型转换，强制类型转换
Cat cat = (Cat)animal;// 编译无错，执行报错!
```

> 这是因为我们在编译的时候进行了强制类型转换，编译时的类型是我们强制转换的类型，所以编译器不会报错，而当我们运行的时候，程序给animal开辟的是Dog类型的内存空间，这与Cat类型内存空间不匹配，所以无法正常转换。这两种情况出错的本质是一样的，所以我们在使用强制类型转换的时候要特别注意这两种错误！！

- 下面有个更安全的方式来实现向下类型转换( instanceof运算符)。 instanceof是Java的一个二元操作符，和==，>，<是同一类。由于它是由字母组成的，所以也是Java的保留关键字。它的作用是测试它左边的对象是否是它右边的类的实例，返回boolean类型数据。

```
/*
1）在强制类型转换中，为了避免出现ClassCastException,可以通过instanceof关键字判断某个引用指向的对象是否为指定类型
2）语法：对象 instanceof 目标类型
3）返回值：
  - true:说明对象就是目标类型或者目标类型的小子类型，可以转换；
  - false:说明不满足强制转换前提，不能转换；
*/

//利用if语句和instanceof运算符来判断两个对象的类型是否一致。
if(animal in instanceof Cat) {
  Cat cat = (Cat)animal;
}
```

- 补充说明：在比较一个对象是否和另一个对象属于同一个类实例的时候，我们通常可以采用instanceof和getClass两种方法通过两者是否相等来判断，但是两者在判断上面是有差别的。Instanceof进行类型检查规则是:你属于该类吗？或者你属于该类的派生类吗？而通过getClass获得类型信息采用==来进行检查是否相等的操作是严格的判断,不会存在继承方面的考虑；

> 总结：在写程序的时候，如果要进行类型转换，我们最好使用instanceof运算符来判断它左边的对象是否是它右边的类的实例，再进行强制转换。

# 八、static与final

## （一）final

- 修饰属性、方法、类
- 修改属性：值不能更改（常量）
- 修饰方法：方法不能重写（继承中子类不能重写父类中被final修饰的成员方法）
- 修饰类：类不能被继承
- 被final修饰的属性：要不显示赋值，要么构造方法赋值

```
//错误：final修饰的属性/变量不能被修改
class Test {
  final int PI = 3.1415926;
  public Test(int pi){
    this.PI = pi;
  }
}
```

- 面试题：
  - 构造方法能不能被final修饰？（不允许）
  - final abstract能不能同时修饰一个类？（不能）

## （二）static

- 修饰：属性、一般方法、游离块、内部类
- 不能修饰：外部类，局部变量、构造方法
- 面试题：能不能修饰外部类和局部变量？（不能）
- 修饰属性：静态属性/类变量

- 修饰方法：静态方法
- java里为什么主函数前面要加static修饰【参考连接】
- static修饰的成员变量不属于对象的数据结构
- static成员变量和类的信息放在一起，仅保留一份，用于共享，而不是在堆中保留在每个对象里的多个副本
- 被static修饰的属性和方法可以用类名字直接调用，可以不创建对象。不过，对象也可以直接调用
- 静态方法里不能使用非静态属性。（eg：main函数不能直接使用本类中的非静态的成员变量）
- static游离块在类加载的就被执行，且只执行一次
- static参与的程序执行次序

  > 父类的静态 子类的静态 父类的非静态 父类的构造 子类的非静态 子类的构造

- 类加载机制

  > .class字节码文件被JVM（Java虚拟机）加载 堆 对象 栈 局部变量 全局数据区 static 代码段

- 静态方法中不能使用super和this(它们实例化的对象)
- static方法的虚方法的调用有什么特点?

  > 虚方法调用对静态方法无效 因为static方法不能被重写

- 面试题

```
class Test {
    // Test example = new Test();
    static Test example = new Test();

    public Test() {
        System.out.println("lalala");
    }

    public static void main(String[] args) {
        Test test = new Test();
    }
}
// 不加static修饰，程序就存在递归
```

# 第六节：设计模式

## 一、问题

1. 什么是设计模式 一种通用方法和手段
2. ：Java中有多少种模式（23种）

- 共23种设计模式。
- 参考资源：《Java与模式》、《Head First设计模式》、菜鸟教程

3. 问题三：设计模式什么场合会用？ 一般在框架里用，eg：spring框架
4. 列举几个Java设计模式

- 工厂模式 ***
- 单例模式 ***
- 适配器模式

- 装饰模式
- 模板方法模式
- 观察者模式
- MVC模式

> 单例模式和工厂模式一定能手写，脱离IDE。 方法：先画类图，再写代码。

# 二、OOP设计原则（参考链接）

- OCP（Open-Closed Principle），开放封闭原则：软件实体应该扩展开放、修改封闭。
- DIP(Dependency Inversion Principle)，依赖倒置原则：摆脱面向过程编程思想中高层模块依赖于低层实现，抽象依赖于具体细节。OOP中要做到的是，高层模块不依赖于低层模块实现，二者都依赖于抽象；抽象不依赖于具体实现细节，细节依赖于抽象。。
- LSP(Liskov Substitution Principle)，Liskov替换原则：继承思想的基础。“只有当衍生类可以替换掉基类，软件单位的功能不会受到影响时，基类才真正被复用，而衍生类也才能够在基类的基础上增加新的行为。”
- ISP（Interface Insolation Principle），接口隔离原则：接口功能单一，避免接口污染。 实现：一个类对另外一个类的依赖性应当是建立在最小的接口上的。使用多个专门的接口比使用单一的总接口要好。
- SRP(Single Resposibility Principle)，单一职责原则：就一个类而言，应该仅有一个引起它变化的原因。 如果一个类的职责过多，就等于把这些职责耦合在一起，一个职责的变化可能会抑止这个类完成其他职责的能力。
- CARP（Composite/Aggregate Reuse Principle），合成/聚合复用原则：设计模式告诉我们对象委托优于类继承，从UML的角度讲，就是关联关系优于继承关系。尽量使用合成/聚合、尽量不使用继承。 实现：在一个新的对象里面使用一些已有的对象，使之成为新对象的一部分，以整合其功能。
- LoD(Law Of Demeter or Principle of Least Knowledge)，迪米特原则或最少知识原则：就是说一个对象应当对其他对象尽可能少的了解。即只直接与朋友通信，或者通过朋友与陌生人通信。

# 三、常用设计模式

## （一）模板方法模式

代码实现

- 写一个抽象类（模板类），有具体的方法
- 创建子类，重写父类的抽象方法
- 面向对象的设计原则 eg：开闭原则

## （二）单例/单子模式（非常重要）

1.解决什么问题？ 一个类只能创建一个对象

2. 什么时候用？ 只需要单一对象的时候，比如一个宿舍只需要一个饮水机
3. 实现模式

- 饿汉式 （1）属性：private、static属性，当前类类型，立即初始化 （2）构造方法必须私有,private修饰 （3）创建get方法，public、static修饰，通过get方法获得当前类的对象（实例） （4）特点：线程安全，效率高

```
public class Person {
    /**属性：private、static修饰，类型为当前类类型，且立即初始化 */
    private static Person person = new Person();

    /**
     * 构造方法必须私有
     */
    private Person() {
        System.out.println(" 这里是饿汉式的私有构造方法! ");
```

```
    }

    /**
     * public、通过get当前类的对象
     * @return
     */
    public static Person getPerson() {
        return person;
    }
}

//测试类
class TestPerson {
    public static void main(String[] args) {
        //static方法，直接通过类调用进行对象初始化
        Person person1 = Person.getPerson();
        Person person2 = Person.getPerson();

        //属于同一个对象
        System.out.println(person1 == person2);
    }
```

- 懒汉式 （1）属性：private、static修饰，类型为当前类类型，不初始化 （2）构造方法必须私有,private修饰 （3）创建get方法，public、static修饰，通过get方法获得当前类的对象（实例），首次获取需要实例化创建对象 （4）特点：线程不安全，效率低 （5）两者区别：对象创建的时间点不同，饿汉式早，懒汉式晚。饿汉是在属性上直接创建对象，懒汉式则是在方法调用的时候再创建对象。

```
public class Student {
    /**属性：private、static修饰，类型为当前类类型，不先初始化，只先声明 */
    private static Student student;

    /**
     * 构造方法必须私有
     */
    private  Student() {
        System.out.println("这里是懒汉模式的私有构造方法！");
    }

     /**
     * public、通过get当前类的对象(首次使用需要实例化)
     * @return student
     */
    public static Student getStudentInstence() {
        if(student == null)
            student = new Student();
        return student;
    }
}

//测试类
class TestStudent {
    public static void main(String[] args) {
                //static方法，直接通过类调用进行对象初始化
        Student stu1 = Student.getStudentInstence();
```

```
        Student stu2 = Student.getStudentInstence();

        //属于同一个对象
        System.out.println(stu1 == stu2);
    }
}
```

### （三）工厂模式

1. 什么是工厂模式？ 向用户屏蔽创建对象的细节，直接取对象即可。比如：我们取手机店买手机，不需要了解手机是如何生产的?
2. 最终目的： 创建者和调用者的分离
3. 分类

- 简单工厂模式
- 工厂方法模式： 定义一个用于创建对象的接口，让子类决定实例化哪一个类，工厂方法是一个类的实例化延迟到其子类。
- 抽象工厂模式：提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类。

4. 使用示例：

- Calendar的getInstance方法：Calendar.getInstance();
- JDBC中Connection对象的获取getConnection方 法:DriverManager.getConnection();
- spring中的IOC容器创建、管理bean对象的时候
- 反射中Class对象的newInstance方法

5. 代码实现

- 简单工厂模式（参见代码部分）
- 工厂方法模式（参见代码部分）

# 第七节：Java API的应用

## 一、基本介绍

### （一）API

1. 什么是API

- API(Application Programming Interface)——应用程序编程接口

> 是一些预先定义的函数，目的是提供应用程序与开发人员基于某软件或硬件得以访问一组例程的能力，而又无需访问源码，或理解内部工作机制的细节。

2. API分类

- JDK带的
- SDK
- 第三方Apache
- 公司自己提供

### （二）微服务

1. 什么是微服务?

- 给我公开接口，我来调用

- 微：小
- 服务：一个功能或者一个系统
- 微服务：小型系统或功能

2. 微服务相关技术
3. Spring Boot
4. Spring Cloud

例：我开发的教务管理系统：教务管理、学历验证、支付、短信验证、学生户籍信息

- 教务管理：开发人员根据需求开发
- 学历验证：调用【学信网】 的接口
- 支付：调用【微信支付】的接口
- 短信验证码：调用【网易】的接口
- 学生户籍验证：调用【公安户籍管理】的接口

# 二、常用API

## （一）String类

1. String类考题

- 面试题 列举几个常用的被final修饰的类 eg:String 、Math、System、Integer等
- 提示：String不是关键字；是不是Java的内置类型？不是。 int String = 90;//正确

2. String类的常用方法

> 多查看java jdk文档

- String(char[] value) //将字符数组转换成字符串
- String newName = new String(name.getBytes("ISO-8859-1"),"UTF-8");// 处理中文乱码
- boolean equals(Object anObject)
- toCharArray(); //将字符串转化成数组
- boolean equals(Object anObject) //区分大小的比价
- boolean equals(Object anObject) //忽略大小写的比较
- string.trim();//去掉字符串的空格
- String substring(int beginIndex, int endIndex) //截取子串
- String substring(int beginIndex) //截取开始索引后的所有字符

```
substring(2);//从索引开始，取到结束
substring(1,1);//有结果，长度为0
substring(4,2);//error
```

- String replace(char oldChar, char newChar) //替换单个字符
- String replace(CharSequence target, CharSequence replacement) //替换多个字符
- char charAt(int index) //返回 char指定索引处的值

## （二）StringBuilder类和StringBuffer类

- String和StringBuilder、StringBuffer区别是什么？ String是不可变字符串类（字符串值/内容不可变，字符串指向可以改变，貌似是一个常量），后者是可变字符串类

```
String str = "chana";
str.toUpperCase();//str还是小写
String str2 = str.toUpperCase();//str2是大写
//str2是大写，str没有改变，还是小写
```

- StringBuilder、StringBuffer是可变字符串类 区别：StringBuffer线程安全的，StringBuilder线程是不安全的(但是效率高)

```

```

- 开发中常用在SQl组合查询上，动态拼接SELECT语句；
- 常用函数

```
append();//追加
delete();//删除
insert();//插入
reverse();//字符串按转
replace();//字符串替换
length();//字符串长度
charAt();//指定索引的字符提取
subString()//专辑顶索引范围字符提取
```

- 面试题:字符串的创建方式有哪些？有什么不同？--- 两种

```
String str1  = "china";
String str2 = new String("china");//创建了几个对象？---2个第一个是:"china",第二个是str2指向的对象（也就是new出来的对象）
String str3 = "china";
```

  - String存储机制---字符串池

## （三）System类

- System.currentTimeMillis()//输出程序的执行时间（毫秒）
- System.nanoTime()//输出程序的执行时间（纳秒）
- arraycopy()//两数组内容替换
- exit()//终止JVM
- gc()//提醒JVM回收垃圾（无用的对象等）

## （四）Math类

- Math.ceil()//相上取整
- Math.floor//向下取整
- Math.random()//产生一个随机[0,1)
- Math.round()//把一个数字舍入为最接近的整数

```
// 结果分析：负数-如果绝对值大于大于0.5，原数+0.5，结果向下取整（变小）
System.out.println(Math.round(-11.2));//-11
System.out.println(Math.round(-11.5));//-11 -11.5+0.5=11
System.out.println(Math.round(-11.6));//-12
System.out.println(Math.round(11.2));//11
System.out.println(Math.round(11.5));//12
System.out.println(Math.round(-0.5));//0
```

## （五）Random类

java.util.Random 是Java随机数类

- 生成0`1000的随机数

```
Random ran = new Random();
int x = ran.next(1000);
```

## （六）Date和Calendar类

```
/**
 * 使用Date类获取当前日期时间
 *
 * @return 当前日期时间
 */
public static String getCurrentDate() {
    Date date = new Date();
    return date.toLocaleString();
}

/**
 * 使用Calendar类获取当前日期时间
 *
 * @return 当前日期时间
 */
public static String getCurrentDateByCalendar() {
    Calendar date = Calendar.getInstance();
    return "" + date.get(Calendar.YEAR) + "-" + (date.get(Calendar.MONTH) + 1) + "-"
            + date.get(Calendar.DAY_OF_MONTH) + " " + date.get(Calendar.HOUR_OF_DAY) + ":"
            + date.get(Calendar.MINUTE) + ":" + date.get(Calendar.SECOND);
}
```

## （七）自定义jar包使用

- 导出包

```
示例 1: 将两个类文件归档到一个名为 classes.jar 的档案中:
       jar cvf classes.jar Foo.class Bar.class
示例 2: 使用现有的清单文件 'mymanifest' 并
           将 foo/ 目录中的所有文件归档到 'classes.jar' 中:
       jar cvfm classes.jar mymanifest -C foo/ .
```

- 导入jar包（Bulid Path）
- 调用jar包中函数

## （八）Object类（顶级父类）

- 位于核心包里
- clone();
- equals();
- toString();
- finalize();

## （九）封装类的使用

- Wrapper class
- 什么是封装类？ 8种基本数据类型对应的类，类型，就是封装类，由Jdk提供，位于Java.long中 boolean--- Boolean byte--- Byte short--- Short char--- Character int--- Integer long--- Long float--- Float double--- Double

## （十）BigDecimal类

- 实现1-0.9的精确计算

```
System.out.println(1-0.9);//0.099999999999
```

- 如何实现精确计算么？---使用java.math.BigDecimal
- 代码示例

```
public class Calc {

    /**
     * 使用BigDecimal实现时间精确计算（但是依然无法计算出）
     *
     * @param num1 计算数字一
     * @param num2 计算数字二
     * @return 不精确计算结果
     */
    public double sub(double num1, double num2) {
        BigDecimal number1 = new BigDecimal(num1);// API参数为double类型
        BigDecimal number2 = new BigDecimal(num2);
        return number1.subtract(number2).doubleValue();
    }

    /**
     * 使用BigDecimal实现时间精确计算（结果正确）
     *
     * @param num1 计算数字一
     * @param num2 计算数字二
     * @return 精确计算结果
     */
    public double sub2(double num1, double num2) {
        BigDecimal number1 = new BigDecimal(num1 + "");// API参数为String类型
        BigDecimal number2 = new BigDecimal(num2 + "");
        return number1.subtract(number2).doubleValue();
    }
```

```
    /**
     * 主方法测试
     * @param args
     */
    public static void main(String[] args) {
        Calc calc = new Calc();
        double num1 = 1;
        double num2 = 0.9;
        // 参数为doule类型
        System.out.println(calc.sub(num1, num2));
        // 参数为String类型
        System.out.println(calc.sub2(num1, num2));
    }
}
```

# 第八节：异常处理

## 一、问题引入：除法计算

- 不使用异常捕获 直接报错，程序终止
- 使用异常捕获（try-catch） 提示友好，程序还可继续执行（跳过try异常后代码）

## 二、异常处理机制如何?

首先：执行到异常时，JVM自动创建一个异常类对象，后面的代码不在执行。然后，catch捕获异常，执行catch中的代码。如果catch无法捕获异常，程序退出。

> Exception 有很多一场子类

## 三、异常处理语法如何?

- try catch finally

```
/**
 * 除法计算（添加异常捕获）
 * @param num1 被除数
 * @param num2 除数
 * @return 相除结果
 */
public double div2(int num1, int num2) {
    double result = 0;
    try {
        //一旦出现异常，JVM直接创建ArithmeticException对象
        result = num1 / num2;
    } catch (Exception e) {
        System.out.println("数据不合法！");
        //e.printStackTrace();
    } finally {// 不管有无异常必定要执行，可选则添加，一般用于释放资源
        System.out.println("finally");
    }
    return result;
}
```

- finally

- 不管有无异常必定要执行
- 可选则添加
- 一般用于释放资源

## 四、手动抛出异常

- 什么时候用？根据编程者的需要，使用`throw`抛出异常

```
/**
 * 除法计算（手动添加异常捕获）
 *
 * @param num1 被除数
 * @param num2 除数
 * @return 相除结果
 */
public double div3(int num1, int num2) {
    double result = 0;
    if (num2 == 0) {
        System.out.println("数据不合法！");
        // 手动抛出异常
        throw new AcceptPendingException();
    } else {
        result = num1 / num2;
    }
    return result;
}
```

## 五、调用者处理异常

- 在方法声明处添加异常，使用`throws`

```
public void read() throws
FileNotFoundException,ArithmeticException {
    FileInputStream fs = new FileInputStream("a.txt");
}
```

- throw和throws的区别？

> throws是用来声明一个方法可能抛出的所有异常信息，throws是将异常声明但是不处理，而是将异常往上传，谁调用我就交给谁处理。 而throw则是指抛出的一个具体的异常类型。

## 六、自定义异常类

- 例如：用户取款时，金额不够。
- 书写规则
  - 继承Exception
  - 两个构造（一个无参，一个带参）

```
/**
 * 模拟取款余额不足时的异常
 * 自定义异常类
 * @author: liuhao
```

```
 * @version 1.0
 */
public class AccountException extends Exception {

    /**
     * 无参构造
     */
    public AccountException() {
        super();
    }

    /**
     * 带参构造
     * @param message
     */
    public AccountException(String message) {
        super(message);
    }

    /**
     * 重写同String
     */
    public String toString() {
        return "AccountException提示：余额不足";
    }

}

//用户账户
class BankAccound {
    public void withDraw() throws AccountException {
        int balance = 200;// 余额
        int withDraw = 500;// 待取金额
        if (withDraw > balance) {
            throw new AccountException("余额不足");
        }
    }

    /**
     * 主方法测试
     * @param args
     */
    public static void main(String[] args) {
        BankAccound bankAccound = new BankAccound();
        try {
            bankAccound.withDraw();
        } catch (AccountException e) {
            e.printStackTrace();
            System.out.println(e);
            System.out.println(e.getMessage());
        }
    }
}
```

## 七、面试题

1. 异常分类

- 已检查异常（Checked Exception）
- 未检查异常/运行时异常（Unchecked Exception）RuntimeException
- 区别：已检查异常在编译是就能发现，未已检查异常执行时才发现

2. 请列举几个常见的异常

- NullPointException 空指针异常
- ArithmeticException 算术计算异常
- ArrayIndexOutOfBoundsException 数组越界异常

3. Exception和error的区别

- 参考链接
- 异常：程序员是可以处理的
- 错误：是未预料到的，由外部环境引起的，一般是JVM崩溃，内存泄露等，程序员处理不了

4. 重写时异常如何处理

- 子类可以重写更多的异常，但这些异常类型必须是父类的子类

# 第九节：Log4j日志

## 一、背景介绍

1. Log4j有两个系列（1.x系列和32.x系列）
2. 官网：

- http://logging.apache.org/log4j/1.2/download.html
- https://logging.apache.org/log4j/2.x/

3. 来自于Apache组织，也是开源的，是一个第三方组件
4. 作用：做日志

## 二、配置及使用

1. 基础配置

- 复制官网下载的jar包，到工程下，并build path添加到构建路径里
- 复制Log4j.properties配置文件到工程src目录下

2. 使用

- 学习网址：https://blog.csdn.net/sinat_30185177/article/details/73550377
- Log4j由三个重要的组件构成：日志信息的优先级，日志信息的输出目的地，日志信息的输出格式。
- 日志信息的优先级：分为OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者您定义的级别。
- 常用优先级：ERROR、WARN、INFO、DEBUG
- 使用流程：
    - 导入jar包 log4j-1.2.16.jar
    - 将配置文件log4j.properties放置在与**src同级目录**下
    - 在代码中使用

```
//得到日志对象
Logger logger = Logger.getLogger(PrintInfo.class);
try {
   new AccountBank().withDrawMethod();
} catch (AccountException e) {
   logger.debug(e);// 调试过程中用debug
   logger.error(e);// 程序运行过程中错误用error
   logger.info(e);// 一般提示信息用info
}
```

## 三、课后

1. 使用Log4j 2.x 做一个工程

- 使用xml,详见码云提交的代码部分
- jar包下载链接
- 配置文件—参考链接

# 第十节：Java I/O

## 一、存放数据的几个地方

- 内存--Redis***【标记】
- 磁盘---文件
- 数据库

## 二、什么是I/O

- input：读文件
- output：写文件

## 三、java.io.File

1. 查看文档
2. 常用函数

- File(String pathname) //构造函数
- boolean exists() //判断文件是否存在
- long length() //文件所占字节长度/大小
- boolean createNewFile() //新建文件
- boolean mkdir() //新建文件夹
- boolean delete() //删除文件/文件夹

> 参考：GreateFile.java

## 四、文件读写问题

1. 按字节读写（二进制文件、文本文件） 读：输入流（FileInputStream） 写：输出流（FileOutputStream）
2. 按字符读写（文本文件） 读：输入流（FileReader） 写：输出流（FileWrite） 有格式文件：POI组件
3. 按行读取 Java.io.buffer
4. JDK8中文件读取方式

- 适用于小的文件

> 参考：FileOperation.java

## 五、对象序列化 和 反序列化

JavaBean EJB--企业级对象

- 序列化目的：将内存对象包含的属性值保存到硬盘上，长期保存对象数据
- 反序列化：将硬盘中存放的数据载入到内存对象中
- 使用场景：两台电脑传输类对象
- 问题：那种类的对象可以序列化？ 如果一个类的对象要被序列化，这个类要实现一个接口：java.io.Serializable---标志接口

  > 标志接口，作用：让一个类的对象可以序列化 eg:Remo 暂态关键字：transient，让不想让序列化的属性不参加序列化，eg:网络上传的敏感信息：密码

> 课后查阅预习：多线程、并发、大并发

# 第十一节：多线程

> Java语言内支持多线程 【参考学习链接】

## 一、创建线程和启动

1. 使用继承方式

- 创建一个Thread子类，继承与Thread类
- 重写父类run()方法
- 创建Thread子类的实例，即创建了线程对象；
- 调用该线程对象的start()方法启动线程。

2. 使用接口方式

- 创建Runnable接口的实现类，实现Runnable接口中的run()方法
- 创建Runnable实现类的实例，并以此实例作为Thread的target对象，即该Thread对象才是真正的线程对象。

3. 通过Callable和Future创建线程

- 创建Callable接口的实现类，并实现call()方法，该call()方法将作为线程执行体，并且有返回值。
- 创建Callable实现类的实例，使用FutureTask类来包装Callable对象，该FutureTask对象封装了该Callable对象的call()方法的返回值。
- 使用FutureTask对象作为Thread对象的target创建并启动新线程。
- 调用FutureTask对象的get()方法来获得子线程执行结束后的返回值其中，Callable接口(也只有一个方法)定义如下：

```
  public interface Callable   {
    V call() throws Exception;
   }

//步骤1：创建实现Callable接口的类SomeCallable(略);
//步骤2：创建一个类对象：
    Callable oneCallable = new SomeCallable();
//步骤3：由Callable创建一个FutureTask对象：
  FutureTask oneTask = new FutureTask(oneCallable);
//注释：  FutureTask是一个包装器，它通过接受Callable来创建，它同时实现了 Future和Runnable接口。
//步骤4：由FutureTask创建一个Thread对象：
  Thread oneThread = new Thread(oneTask);
```

```
//步骤5：启动线程：
  oneThread.start();
```

# 二、线程的生命周期

1. 新建状态

- 用new关键字和Thread类或其子类建立一个线程对象后，该线程对象就处于新生状态。处于新生状态的线程有自己的内存空间，通过调用start方法进入就绪状态（runnable）。
- 注意：不能对已经启动的线程再次调用start()方法，否则会出现java.lang.IllegalThreadStateException异常。

2. 就绪状态

- 处于就绪状态的线程已经具备了运行条件，但还没有分配到CPU，处于线程就绪队列（尽管是采用队列形式，事实上，把它称为可运行池而不是可运行队列。因为cpu的调度不一定是按照先进先出的顺序来调度的），等待系统为其分配CPU。等待状态并不是执行状态，当系统选定一个等待执行的Thread对象后，它就会从等待执行状态进入执行状态，系统挑选的动作称之为“cpu调度”。一旦获得CPU，线程就进入运行状态并自动调用自己的run方法。
- 提示：如果希望子线程调用start()方法后立即执行，可以使用Thread.sleep()方式使主线程睡眠一会儿，转去执行子线程。

3. 运行状态

- 处于运行状态的线程最为复杂，它可以变为阻塞状态、就绪状态和死亡状态。
- 处于就绪状态的线程，如果获得了cpu的调度，就会从就绪状态变为运行状态，执行run()方法中的任务。如果该线程失去了cpu资源，就会又从运行状态变为就绪状态。重新等待系统分配资源。也可以对在运行状态的线程调用yield()方法，它就会让出cpu资源，再次变为就绪状态。
- 注： 当发生如下情况是，线程会从运行状态变为阻塞状态：
    - 线程调用sleep方法主动放弃所占用的系统资源
    - 线程调用一个阻塞式IO方法，在该方法返回之前，该线程被阻塞
    - 线程试图获得一个同步监视器，但更改同步监视器正被其他线程所持有
    - 线程在等待某个通知（notify）
    - 程序调用了线程的suspend方法将线程挂起。不过该方法容易导致死锁，所以程序应该尽量避免使用该方法，目前以不推荐使用。
- 当线程的run()方法执行完，或者被强制性地终止，例如出现异常，或者调用了stop()、desyory()方法等等，就会从运行状态转变为死亡状态。

4. 阻塞状态

- 处于运行状态的线程在某些情况下，如执行了sleep（睡眠）方法，或等待I/O设备等资源，将让出CPU并暂时停止自己的运行，进入阻塞状态。
- 在阻塞状态的线程不能进入就绪队列。只有当引起阻塞的原因消除时，如睡眠时间已到，或等待的I/O设备空闲下来，线程便转入就绪状态，重新到就绪队列中排队等待，被系统选中后从原来停止的位置开始继续运行。

5. 死亡状态

- 当线程的run()方法执行完，或者被强制性地终止，就认为它死去。这个线程对象也许是活的，但是，它已经不是一个单独执行的线程。线程一旦死亡，就不能复生。 如果在一个死去的线程上调用start()方法，会抛出java.lang.IllegalThreadStateException异常。

# 三、线程管理

1. 线程睡眠——sleep()

- 如果我们需要让当前正在执行的线程暂停一段时间，并进入阻塞状态，则可以通过调用Thread的sleep方法。

- sleep是静态方法，最好不要用Thread的实例对象调用它,因为它睡眠的始终是当前正在运行的线程，而不是调用它的线程对象，它只对正在运行状态的线程对象有效。如下面的例子：

```
public class Test1 {
    public static void main(String[] args) throws InterruptedException {
        System.out.println(Thread.currentThread().getName());
        MyThread myThread=new MyThread();
        myThread.start();
        myThread.sleep(1000);//这里sleep的就是main线程，而非myThread线程
        Thread.sleep(10);
        for(int i=0;i<100;i++){
            System.out.println("main"+i);
        }
    }
}
```

- Java线程调度是Java多线程的核心，只有良好的调度，才能充分发挥系统的性能，提高程序的执行效率。但是不管程序员怎么编写调度，只能最大限度的影响线程执行的次序，而不能做到精准控制。因为使用sleep方法之后，线程是进入阻塞状态的，只有当睡眠的时间结束，才会重新进入到就绪状态，而就绪状态进入到运行状态，是由系统控制的，我们不可能精准的去干涉它，所以如果调用Thread.sleep(1000)使得线程睡眠1秒，可能结果会大于1秒。

2. 线程让步——yield()

- yield()方法和sleep()方法有点相似，它也是Thread类提供的一个静态的方法，它也可以让当前正在执行的线程暂停，让出cpu资源给其他的线程。**但是和sleep()方法不同的是，它不会进入到阻塞状态，而是进入到就绪状态。** yield()方法只是让当前线程暂停一下，重新进入就绪的线程池中，让系统的线程调度器重新调度器重新调度一次，完全可能出现这样的情况：当某个线程调用yield()方法之后，线程调度器又将其调度出来重新进入到运行状态执行。
- 实际上，当某个线程调用了yield()方法暂停之后，优先级与当前线程相同，或者优先级比当前线程更高的就绪状态的线程更有可能获得执行的机会，当然，只是有可能，因为我们不可能精确的干涉cpu调度线程。用法如下：

```
public class Test1 {
    public static void main(String[] args) throws InterruptedException {
        new MyThread("低级", 1).start();
        new MyThread("中级", 5).start();
        new MyThread("高级", 10).start();
    }
}

class MyThread extends Thread {
    public MyThread(String name, int pro) {
        super(name);// 设置线程的名称
        this.setPriority(pro);// 设置优先级
    }

    @Override
    public void run() {
        for (int i = 0; i < 30; i++) {
            System.out.println(this.getName() + "线程第" + i + "次执行! ");
            if (i % 5 == 0)
                Thread.yield();
        }
    }
}
```

- 关于sleep()方法和yield()方的区别如下：
    - sleep方法暂停当前线程后，会进入阻塞状态，只有当睡眠时间到了，才会转入就绪状态。而yield方法调用后，是直接进入就绪状态，所以有可能刚进入就绪状态，又被调度到运行状态。
    - sleep方法声明抛出了InterruptedException，所以调用sleep方法的时候要捕获该异常，或者显示声明抛出该异常。而yield方法则没有声明抛出任务异常。
    - sleep方法比yield方法有更好的可移植性，通常不要依靠yield方法来控制并发线程的执行。

3. 线程合并——join()

- 线程的合并的含义就是将几个并行线程的线程合并为一个单线程执行，应用场景是当一个线程必须等待另一个线程执行完毕才能执行时，Thread类提供了join方法来完成这个功能。**注意:它不是静态方法。**
- 它有3个重载的方法：

```
void join()
     //当前线程加入该线程后面，等待该线程终止。
void join(long millis)
     //当前线程等待该线程终止的时间最长为 millis 毫秒。 如果在millis时间内，该线程没有执行完，那么当前线程进入就绪状态，重新等待cpu调度
void join(long millis,int nanos)
     //等待该线程终止的时间最长为 millis 毫秒 + nanos 纳秒。如果在millis时间内，该线程没有执行完，那么当前线程进入就绪状态，重新等待cpu调度
```

4. 设置线程的优先级

- 每个线程执行时都有一个优先级的属性，优先级高的线程可以获得较多的执行机会，而优先级低的线程则获得较少的执行机会。与线程休眠类似，线程的优先级仍然无法保障线程的执行次序。只不过，优先级高的线程获取CPU资源的概率较大，优先级低的也并非没机会执行。
- 每个线程默认的优先级都与创建它的父线程具有相同的优先级，在默认情况下，main线程具有普通优先级。
- 注：Thread类提供了setPriority(int newPriority)和getPriority()方法来设置和返回一个指定线程的优先级，其中setPriority方法的参数是一个整数，范围是1~10之间，也可以使用Thread类提供的三个静态常量：

```
MAX_PRIORITY  = 10

MIN_PRIORITY  = 1

NORM_PRIORITY  = 5
```

> 注：虽然Java提供了10个优先级别，但这些优先级别需要操作系统的支持。不同的操作系统的优先级并不相同，而且也不能很好的和Java的10个优先级别对应。所以我们应该使用MAX_PRIORITY、MIN_PRIORITY和NORM_PRIORITY三个静态常量来设定优先级，这样才能保证程序最好的可移植性。

```
public class Test1 {
      public static void main(String[] args) throws InterruptedException {
          new MyThread("高级", 10).start();
          new MyThread("低级", 1).start();
      }
  }

  class MyThread extends Thread {
      public MyThread(String name,int pro) {
          super(name);//设置线程的名称
          setPriority(pro);//设置线程的优先级
```

```
    }
    @Override
    public void run() {
        for (int i = 0; i < 100; i++) {
            System.out.println(this.getName() + "线程第" + i + "次执行! ");
        }
    }
}
```

5. 后台（守护）线程

- 守护线程使用的情况较少，但并非无用，举例来说，JVM的垃圾回收、内存管理等线程都是守护线程。还有就是在做数据库应用时候，使用的数据库连接池，连接池本身也包含着很多后台线程，监控连接个数、超时时间、状态等等。调用线程对象的方法setDaemon(true)，则可以将其设置为守护线程。
- 守护线程的用途为：
  - 守护线程通常用于执行一些后台作业，例如在你的应用程序运行时播放背景音乐，在文字编辑器里做自动语法检查、自动保存等功能。
  - Java的垃圾回收也是一个守护线程。守护线程的好处就是你不需要关心它的结束问题。例如你在你的应用程序运行的时候希望播放背景音乐，如果将这个播放背景音乐的线程设定为非守护线程，那么在用户请求退出的时候，不仅要退出主线程，还要通知播放背景音乐的线程退出；如果设定为守护线程则不需要了。
- setDaemon方法的详细说明：

```
/**
 * 将该线程标记为守护线程或用户线程。当正在运行的线程都是守护线程时，Java 虚拟机退出。
 * 该方法必须在启动线程前调用。 该方法首先调用该线程的 checkAccess 方法，且不带任何参数。这可能抛出
SecurityException（在当前线程中）。
 * 参数：
 *       on - 如果为 true，则将该线程标记为守护线程。
 * 抛出：
 *      IllegalThreadStateException - 如果该线程处于活动状态。
 *      SecurityException - 如果当前线程无法修改该线程。
 * @param on
 */
public final void setDaemon(boolean on)
```

- 注：JRE判断程序是否执行结束的标准是所有的前台执线程行完毕了，而不管后台线程的状态，因此，在使用后台县城时候一定要注意这个问题。

6. 正确结束线程

- Thread.stop()、Thread.suspend、Thread.resume、Runtime.runFinalizersOnExit这些终止线程运行的方法已经被废弃了，使用它们是极端不安全的！
- 想要安全有效的结束一个线程，可以使用下面的方法：
  - 正常执行完run方法，然后结束掉；
  - 控制循环条件和判断条件的标识符来结束掉线程。

```
class MyThread extends Thread {
    int i=0;
    boolean next=true;

    @Override
    public void run() {
```

```
        while (next) {
            if(i==10)
                next=false;
            i++;
            System.out.println(i);
        }
    }
}
```

# 四、线程同步、共享数据

> java允许多线程并发控制，当多个线程同时操作一个可共享的资源变量 / 临界资源时（如数据的增删改查），将会导致数据不准确，相互之间产生冲突，因此加入同步锁以避免在该线程没有完成操作之前，被其他线程的调用，从而保证了该变量的唯一性和准确性

1. 什么是同步、异步?

- 同步：线程执行有先后次序，一个线程结束才能执行另一个。同步保障了线程的安全但过多地占用CPU资源使得CPU利用率降低
- 异步：一哄而上

2. 解决线程同步的方法

- 同步方法： 使用 synchronized 关键字修饰方法

```
public synchronized void withDraw(double count) {
    System.out.println(Thread.currentThread().getName() + ": 进入取款方法...");
    if (count <= this.balance) {
        System.out.println(Thread.currentThread().getName() + ": 正在取款...");
        this.balance -= count;
        System.out.println(Thread.currentThread().getName() + ": 取款成功! 剩余金额: " +
this.balance);
    } else {
        System.out.println(Thread.currentThread().getName() + ": 余额不足，请及时充值...");
    }
}
```

- 同步代码块：使用 synchronized 关键字修饰语句块

```
public void withDraw(double count) {
    System.out.println(Thread.currentThread().getName() + ": 进入取款方法...");
    synchronized (this) {
        if (count <= this.balance) {
            System.out.println(Thread.currentThread().getName() + ": 正在取款...");
            this.balance -= count;
            System.out.println(Thread.currentThread().getName() + ": 取款成功! 剩余金额: " +
this.balance);
        } else {
            System.out.println(Thread.currentThread().getName() + ": 余额不足，请及时充值...");
        }
    }
}
```

- 注：同步是一种高开销的操作，因此应该尽量减少同步的内容。通常没有必要同步整个方法，使用synchronized代码块同步关键代码即可。

3. 实例：账户取款问题

> 一个账户，两人同时取款（两个线程共享同一个账户）

- 实现：参考 `ThreadsSharedata.java`

## 五、线程通信

1. 借助于Object类的wait()、notify()和notifyAll()实现通信

- wait()：线程执行wait()后，就放弃了运行资格，处于冻结状态；
- notify()：线程运行时，内存中会建立一个线程池，冻结状态的线程都存在于线程池中，notify()执行时唤醒的也是线程池中的线程，线程池中有多个线程时唤醒第一个被冻结的线程。
- notifyall()：唤醒线程池中所有线程。

2. 注：

- wait(), notify(),notifyall()都用在同步里面，因为这3个函数是对持有锁的线程进行操作，而只有同步才有锁，所以要使用在同步中；
- wait(),notify(),notifyall(), 在使用时必须标识它们所操作的线程持有的锁，因为等待和唤醒必须是同一锁下的线程；而锁可以是任意对象，所以这3个方法都是Object类中的方法。

3. 实例：生产者消费者问题

- 问题描述：生产者——消费者问题是指有两组线程共享一个环形的缓冲池，一组称为生产者，一组称为消费者。缓冲池是由若干个大小相等的缓冲区组成，每个缓冲区可以容纳一个产品。生产者线程不断的将产品放入缓冲池中，消费者不断将产品从缓冲池中取出。
- 核心：生产者——消费者问题，既存在着线程程同步问题，也存在着临界区互斥问题。当缓冲区满时，表示供大于求，生产者必须停止生产，进入等待状态，同时唤醒消费者；当所有缓冲区都为空时，表示供不应求，消费者必须停止消费，唤醒生产者。这就是生产者线程和消费者线程的同步关系。对于缓冲池，生产者和消费者都要使用它，显然他是一个临界资源，对于缓冲池的操作必须是互斥的。
- 实现：参考 `ProducerConsumer.java`

## 六、理解并口述

1. 程序、进程、线程之间的关系

> 【参考链接】

- 程序：程序只是一组指令的有序集合,本身并没有任何运行的含义，它只是一个静态的实体。
- 进程：系统资源分配和调度的基本单位（独占系统资源），它是程序在某个数据集上的执行，有一定的生命周期，反映了一个程序在一定的数据集上运行的全部动态过程。进程之间支持并发执行。
- 线程：线程是进程的一个实体,CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位，共享资源，线程之间可以并发执行 eg：迅雷下载
- 一个正在执行的程序至少有一个进程,一个进程至少有一个线程.

2. 实现线程的方式

- 通过继承Thread类
- 通过实现Runnable接口
- 通过 Callable 和 Future 创建线程。

3. 线程的状态

- 创建状态、就绪状态、运行状态、阻塞状态、消亡状态

4. 线程同步机制

- 锁机制/信号量机制
- 如果同步，那个线程获得锁，那个线程才能执行

5. 同步的目的

- 防止多线程共享数据出现问题

6. 线程方法和对象方法

- 线程方法：来自于Thread的方法eg：start()、run()、sleep()、yield()
- 对象方法：来自于Object的方法eg：wait()、notify() 、notifyAll()

7. 结束线程

- 方式一：自然结束，run()运行结束
- 方式二：方法中发生异常

8. 什么时候使用线程同步

- 资源/数据被多个线程共享使用时

9. 什么是并发?

- 一段时间内，允许多个进程/线程同时运行（宏观上讲）

10. 面试题：sleep()和wait()和yield()和join()

- 线程睡眠-sleep()：静态方法，让当前正在执行的线程暂停一段时间，进入到阻塞状态,不释放锁,可以在任何地方使用；
- wait()：释放锁，进入等待队列，待调用notify()/notifyAll()唤醒指定的线程或者所有线程，才会进入锁池,只能在同步方法或同步块中使用；
- 线程让步-yield()：不释放锁，也不会进入到阻塞状态，而是进入到就绪状态。yield()方法只是让当前线程暂停一下，重新进入就绪的线程池中，让系统的线程调度器重新调度器重新调度一次（按照优先级）。
- 线程合并-join()：线程的合并的含义就是将几个并行线程的线程合并为一个单线程执行，应用场景是当一个线程必须等待另一个线程执行完毕才能执行时，Thread类提供了join方法来完成这个功能，注意，它不是静态方法。

# 第十二节：Collection（Java集合）

## 一、背景

1. 什么东西可以容纳多个对象？数组、集合

- 数组中放的对象是固定的（动态的例外），且对象类型单一
- 集合里的对象个数是动态的，且可以放任何类型对象

2. Java 集合框架

- 参考链接

3. Collection位于Java.util包中，是一个接口
4. 开发上用的最多的是：List、Set、Map

- 注：Map和Collection没有构成继承关系
- 分为两大类：List、Set 和 Map

## 二、List

1. add() 向集合里添加对象

```
List list = new ArrayList();
list.add(12);
list.add("hello");
```

- List list = new ArrayList();//默认初始容量为10
- List list = new ArrayList(15);

2. 泛型，指定集合可以放入的对象

- 方便从集合中获取对象
- size()得到集合内对象的实际个数
- List有两种遍历方式
- List是有序列表，可重复

```
/** 使用泛型 */
List<String> strList = new ArrayList<String>();
strList.add(12);//报错
strList.add("hello");

/**集合和遍历的第一种方式*/
for (int i = 0; i < strList.size(); i++) {
    System.out.println(strList.get(i));
}
/**集合和遍历的第二种方式*/
for( String str : strList) {
    System.out.println(str);
}
/**集合和遍历的第三种方式:: 使用迭代器*/
Iterator iter = null;
for (iter = strList.iterator(); iter.hasNext();) {
    String str = (String)iter.next();
    System.out.println(str);
}
```

3. 去掉List中重复对象

- 重写hashCode和equals方法
- 用Set集合包裹List，使用构造方法
- 常用在：Hibernate中

```
public static void main(String[] args) {
    Student stu1 = new Student("张三",21,"2016");
    Student stu2 = new Student("李四",21,"2015");
    Student stu3 = new Student("张三",21,"2016");

    List<Student> stuList = new ArrayList<Student>();
    stuList.add(stu1);
    stuList.add(stu2);
```

```
    stuList.add(stu3);
    System.out.println(stuList.size());

    /** 包装List */
    Set<Student> setList = new HashSet<Student>(stuList);
    System.out.println(setList.size());
}
```

# 三、Set

- Set没有索引方法
- Set里的对象是无序且不可重复的

```
Set<String> set = new HashSet<String>();
set.add("Hello");
set.add("world");
set.add("!");
set.add("Hello");
System.out.println(set.size());
for(String str : set) {
    System.out.println(str);
}
/* 结果:
  3
  !
  world
  Hello
*/
```

- List和Set的区别
  - List有序可重复，Set无序且不可重复
  - List两种遍历方法，Set一种

# 四、Map

- Map由键值对构成
- 常用方法
  - put(K,V)
  - size()
  - get(K)
- Map取值：按值取值

```
Map Map = new HashMap();
Map.put("学号", "2016");// 键-值对
Map.put("姓名", "张三");
Map.put("性别", "男");
Map.put("性别", "女");
System.out.println(Map.size());

/** Map取值：按名取值 */
System.out.println(Map.get("性别"));
System.out.println(Map.get("学号"));
```

## 五、LinkedList

- 用法：参考链接
- java的数据结构
- 来自java.util包

## 六、Stack--(类)

- peek：栈顶指针
- push：进栈
- pop：出栈
- 先进后出

## 七、Queue--(接口)

- 队列，先进先出

## 八、面试题

1. List、Set、Map的区别

- List和Set继承自Collection接口，Map是独立的接口
- Map中存放的是键值对，List和Set存放值
- Set无序不可重复，List有序可重复

2. Collection和Collections区别

- Collection是集合接口，提供了对集合对象进行基本操作的通用接口方法
- Collections是工具类，提供了一系列的静态方法来操作集合
- 类比：Array.sort()

3. 什么是泛型

- 参考链接
- 泛型，本质是“参数化类型”，通俗的讲就是操作的数据类型被指定为一个参数，这种参数类型可以用在类、接口和方法的创建中，分别称为泛型类、泛型接口、泛型方法。
- Java语言引入泛型的好处是安全简单，在编译的时候检查类型安全，并且所有的强制转换都是自动和隐式的，以提高代码的重用率。
- 在面向对象编程及各种设计模式中有非常广泛的应用
- eg：

```
List<String> strList = new ArrayList<String>();
```

4. ArrayList、Hashtable、HashSet、TreeSet、LinkedHashSet

- ArrayList
  - 初始容量是10
- ArrayList和Vector的区别?
  - Vector类所有方法都是同步的，保证线程安全；而ArrayList不同步；
  - ArrayList内存不够时默认是扩展50% ， Vector是默认扩展100%
- Hashtable:
  - 线程安全,底层由数组+链表实现
  - key，value不能为null
  - 初始容量11， 扩容方式 *2+1
- HashMap:
  - 非线程安全
  - key, value可以为null
  - 初始容量16， 扩容方式 *2
- ConcurrentHashMap
  - 线程安全的
  - 效率比Hashable效率高
- HashSet
  - implements Set
  - 非同步，非线程安全
  - 无序存放，不存放重复元素，可以有null元素
  - 哈希表结构实现
- TreeSet
  - implements SortedSet
  - 非同步
  - 有序存放，不接受重复元素，不可以有null元素
  - 红黑树结构实现

```
public static void main(String[] args) {
    TreeSet<String> treeSet = new TreeSet<String>();
    treeSet.add("hello");
    treeSet.add("world");
    treeSet.add("hello");
    System.out.println(treeSet.size());

    for(String str:treeSet) {
        System.out.println(str);
    }
}
/*
    结果:
    2
    hello
    world
*/
```

- LinkedHashSet:(extends HashSet)
    - 非同步
    - 可以有一个null元素，不接受重复元素，元素按照放入的顺序排列
    - 哈希表+链表实现，链表结构保证了有序

# 第十三节：其它

## 一、Java 网络相关知识

### (一) Ip

- www.tit.edu.cn -- 180.96.16.231
- .com/.com.cn/.cn
- IPV4与IPV6
- 配置IP:cmd + ncpa.cpl
- 公网访问自己开发的系统
    1. 买个远程主机（服务器），阿里云、腾讯
    2. 把我们开发的软件部署到服务器上
    3. 买一个域名，与IP进行绑定，就是解析
    4. 通过域名访问的自己的系统

### (二) TCP/IP、UDP

1. TCP：传输控制协议，特点：面向链接的，可靠的
2. UDP：用户数据报协议，特点：无连接、不可靠、快速传输
3. TCP/IP三次握手：参考链接

### (三) 网络的七层模型

> 七层模型，亦称OSI（Open System Interconnection）。参考模型是国际标准化组织（ISO）制定的一个用于计算机或通信系统间互联的标准体系，一般称为OSI参考模型或七层模型。

- 应用层 网络服务与最终用户的一个接口。 协议有：HTTP FTP TFTP SMTP SNMP DNS TELNET HTTPS POP3 DHCP
- 表示层 数据的表示、安全、压缩。（在五层模型里面已经合并到了应用层） 格式有，JPEG、ASCll、DECOIC、加密格式等
- 会话层 建立、管理、终止会话。（在五层模型里面已经合并到了应用层） 对应主机进程，指本地主机与远程主机正在进行的会话
- 传输层 定义传输数据的协议端口号，以及流控和差错校验。 协议有：TCP UDP，数据包一旦离开网卡即进入网络传输层
- 网络层 进行逻辑地址寻址，实现不同网络之间的路径选择。 协议有：ICMP IGMP IP（IPV4 IPV6） ARP RARP
- 数据链路层 建立逻辑连接、进行硬件地址寻址、差错校验 [2] 等功能。（由底层网络定义协议） 将比特组合成字节进而组合成帧，用MAC地址访问介质，错误发现但不能纠正。
- 物理层 建立、维护、断开物理连接。（由底层网络定义协议）

### (四) Scoket套接字

- 套接字 = IP地址 + 端口号
- 解决客户端的（C/S）结构的软件

## 二、Java注解

- 英文：nnotation
- 注释和注解的区别?
  - 注解 ： 参与代码编译，以@开头的。它是给应用程序看的，单独使用注解毫无意义，一定 要跟工具一起使用,这个所谓的工具实际就是能读懂注解的应用程序 。
  - 注释 ： 对代码没有影响。对代码起到解释、说明的作用;
- 用于配置，比如：Servlet配置、Spring MVC 、Spring、Mybatis
- 可自定义注解，一种应用就是为类提供属性值

## 三、enum—枚举

- 参考链接
- 用来替代常量接口

```
public enum Color {
  red, green, blue;
}
```

## 四、内部类

- 内部类分类:
  - 局部内部类 不写权限修饰符; 可以使用final和abstract
  - 成员内部类 权限修饰符都适用；可以使用final、abstract和static
  - 匿名内部类 使用场景：如果接口的实现或者父类的子类只需要使用唯一的一次
- 主要用于GUI编程
- 如何创建内部类对象?

  ```
  //非静态
  Outer.Inner inner = new Outer().new Inner();
  //静态
  Outer.Inner2 inner2 = new Outer().Inner();
  ```

- 一个类中可以出现哪些内容?
  - 属性
  - 构造方法
  - 一般方法
  - 游离块
  - 内部类
  - 内部接口

## 五、GUI编程--图形用户界面

- GUI---图形用户界面
- java.awt
- java.swing
- 开发：代码书写/使用WindowBuilder插件

# 第四章 数据库操作

## 第一节：JDBC

### 一、JDBC背景介绍

#### （一）什么是JDBC?

- JDBC（Java DataBase Connectivity，java数据库连接）是一种用于执行SQL语句的Java API，可以为多种关系数据库提供统一访问，它由一组用[Java语言]编写的类和接口组成。

#### （二）JDBC作用是什么?

- 与数据库建立连接、发送 操作数据库的语句并处理结果
- 架起Java与数据库之间的桥梁

#### （三）主要的JDBC API有哪些?

1. 接口

- [Connection]：close()、commit()、createStatement()、getAutoCommit()、isClosed()、prepareCall(String sql)、prepareStatement(String sql)、rollback()、setAutoCommit(boolean autoCommit)
- [PreparedStatement]：addBatch()、executeQuery()、executeUpdate()、setInt(int parameterIndex, int x) 等一些列set方法、
- [ResultSet]-结果集：absolute(int row)、afterLast()、beforeFirst()、close() ResultSet释、first()、一些列get方法、
- [Statement]：开发基本不用，但要清楚

2. 类

- [DriveManager]（驱动程序管理器）：getConnection(String url) 及重载方法

3. 异常

- [SQLException]：

#### （四）JDBC数据操作的步骤?

1. 注册驱动
2. 获取链接
3. 定义SQL语句
4. 准备SQL语句
5. 占位符赋值（可能没有）
6. 执行SQL语句
7. 处理执行结果
8. 关闭链接

### 二、实践操作

#### （一）启动Oracle

1. 启动服务

2. 开启数据库管理工具：sqldeveloper，连接数据库
3. 建表

## (二)新建一个动态web项目

- DAO：(Data Access Object) 数据访问对象，数据库操作的代码
- 流程
  - JSP(AJAS) -- Servlet -- Service -- DAO -- DB
  - 显示层/视图层（JSP/HTML）-- 控制层(Servlet) -- 业务处理层(Servicr) -- 数据访问层(Dao) -- 数据源(数据库)
- 需要的外部包
  - Log4j：日志
  - JUnit：单元测试
  - Oracle驱动（D:\Software\app\product\11.2.0\dbhome_1\jdbc\lib\ojdbc6.jar）

## (三）链接数据库

1. Oracle两种连接数据库方式 oci 和 thin

- **oci**

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con = DriverManager.getConnection("jdbc:oracle:oci:@数据库名","登录名", "密码");
```

- **thin**

```
Class.forName("oracle.jdbc.OracleDriver");// 注册Oracle驱动,
String url = "jdbc:oracle:thin:@localhost:1521:orcl";
// jdbc 数据库协议主协议 thin连接方式 @固定语法 localhost为ip地址 1521为端口号    orcl是Oracle数据库驱动的名称
Connection conn = DriverManager.getConnection(url, "scott", "scott");// 获得数据库连接
```

- 区别

> 1）从使用上来说，oci必须在客户机上安装oracle客户端或才能连接，而thin就不需要，因此从使用上来讲thin还是更加方便，这也是thin比较常见的原因。 2）原理上来看，thin是纯java实现tcp/ip的c/s通讯；而oci方式，客户端通过native java method调用c library访问服务端，而这个c library就是oci(oracle called interface)，因此这个oci总是需要随着oracle客户端安装（从oracle10.1.0开始，单独提供OCI Instant Client，不用再完整的安装client） 3）它们分别是不同的驱动类别，oci是二类驱动， thin是四类驱动，但它们在功能上并无差异。 4) 如果想要高性能，请使用OCI连接，如果不想装Oracle客户端，请使用thin连接。 5) oci不需要写IP thin需要

2. 链接数据库代码

```
public class DBUtil {
    /**
     * 获取Oracle数据库链接
     *
     * @return 数据库的链接对象
     */
    public static Connection getConnection() {
        /** 定义一个链接对象，Connection是一个接口 */
        Connection conn = null;
        try {
            /** 注册Oracle驱动，参数是Oracle数据库驱动的名称 */
```

```
            Class.forName("oracle.jdbc.OracleDriver");
            /** 链接字符串（主协议、子协议、链接方式） */
            //jdbc 数据库协议主协议
            //thin连接方式
            //@固定语法
            //localhost为ip地址
            //1521为端口号
            //orcl是Oracle数据库驱动的名称
            String url = "jdbc:oracle:thin:@localhost:1521:orcl";
            /** 获得数据库连接 */
            conn = DriverManager.getConnection(url, "scott", "scott");
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }

    /**
     * 关闭连接
     *
     * @param conn  链接对象
     * @param pstmt 预编译对象
     * @param rs    结果集对象
     */
    public static void close(ResultSet rs, PreparedStatement pstmt, Connection conn) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /*
    * 测试是否可以链接到数据库
    */
    public static void main(String[] args) {
        System.out.println(DBUtil.getConnection());
    }
}
```

## (四) 添删改查（CRUD）

- 详细请参照上传程序
- 例子

```
/** 检查登陆 */
public boolean checkUser(String userName, String userPwd) {
    /** 登陆状态：true--成功，false--失败 */
    boolean result = false;
    /** 连接数据库 */
    Connection conn = null;
    /** 预编译对象 */
    PreparedStatement pstmt = null;
    /** 结果集 */
    ResultSet rs = null;
    /** 获取连接 */
    conn = DBUtil.getConnection();
    /** 定义SQL语句 */
    String checkUserSQL = "SELECT * FROM uol_user WHERE username = ? AND userpwd = ?";

    try {
        /** 准备SQL语句/创建一个执行SQL语句的对象 */
        pstmt = conn.prepareStatement(checkUserSQL);
        /** 给参数?赋值 */
        pstmt.setString(1, userName);// 给第一个参数?赋值
        pstmt.setString(2, userPwd);// 给第二个参数?赋值
        /** 执行SQL语句 */
        rs = pstmt.executeQuery();// 执行查询
        // int result = pstmt.executeUpdate();// 执行更新（添加、删除、修改）

        /** 处理执行结果 */
        if (rs.next()) {
            result = true;
        } else {
            result = false;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        DBUtil.close(rs, pstmt, conn);
    }
    return result;
}
```

# 第二节：链接池

## 一、背景介绍

> 实际开发中“获得连接”或“释放资源”是非常消耗系统资源的两个过程，为了解决此类性能问题（并发时），通常情况我们采取连接池技术，来共享连接Connection。在程序开始的时候，可以创建几个连接，将连接放入到连接池中。用户使用连接的时候，可以从连接池中进行获取。用完之后，可以调用 close() 方法将连接[归还]连接池。

### （一）概念

- 用池来管理Connection，这样可以重复使用Connection。有了池，我们就不用自己来创建Connection，而是通过池来获取Connection对象。当使用完Connection后，调用Connection的close()方法也不会真的关闭Connection，而是把Connection“归还”给池。池就可以再利用这个Connection对象了。

### （二） 规范

- Java为数据库连接池提供了公共的接口，javax.sql.DataSource，各个厂商需要让自己的连接池实现这个接口。这样应用程序可以方便地切换不同厂商的连接池。

## （三）为什么使用连接池

1. 对于一个简单的数据库应用，由于对于数据库的访问不是很频繁。这时可以简单地在需要访问数据库时，就新创建一个连接，用完后就关闭它，这样做也不会带来什么明显的性能上的开销。但是对于一个复杂的数据库应用，情况就完全不同了。频繁的建立、关闭连接，会极大的减低系统的性能，因为对于连接的使用成了系统性能的瓶颈。
2. 连接复用。通过建立一个数据库连接池以及一套连接使用管理策略，使得一个数据库连接可以得到高效、安全的复用，避免了数据库连接频繁建立、关闭的开销。
3. 对于共享资源，有一个很著名的设计模式：资源池。该模式正是为了解决资源频繁分配、释放所造成的问题的。把该模式应用到数据库连接管理领域，就是建立一个数据库连接池，提供一套高效的连接分配、使用策略，最终目标是实现连接的高效、安全的复用。

## （四） 数据库连接池技术带来的优势

1. 资源重用 由于数据库连接得到重用，避免了频繁创建、释放连接引起的大量性能开销。在减少系统消耗的基础上，另一方面也增进了系统运行环境的平稳性（减少内存碎片以及数据库临时进程/线程的数量）。
2. 更快的系统响应速度 数据库连接池在初始化过程中，往往已经创建了若干数据库连接置于池中备用。此时连接的初始化工作均已完成。对于业务请求处理而言，直接利用现有可用连接，避免了数据库连接初始化和释放过程的时间开销，从而缩减了系统整体响应时间。
3. 新的资源分配手段 对于多应用共享同一数据库的系统而言，可在应用层通过数据库连接的配置，实现数据库连接池技术，几年钱也许还是个新鲜话题，对于目前的业务系统而言，如果设计中还没有考虑到连接池的应用，那么.....快在设计文档中加上这部分的内容吧。某一应用最大可用数据库连接数的限制，避免某一应用独占所有数据库资源。
4. 统一的连接管理，避免数据库连接泄漏 较为完备的数据库连接池实现中，可根据预先的连接占用超时设定，强制收回被占用连接。从而避免了常规数据库连接操作中可能出现的资源泄漏。

## （五）连接池的种类

参考链接

1. [Druid] 第三方组件（阿里）

- Druid是Java语言中最好的数据库连接池，Druid能够提供强大的监控和扩展功能，是一个可用于大数据实时查询和分析的高容错、高性能的开源分布式系统，尤其是当发生代码部署、机器故障以及其他产品系统遇到宕机等情况时，Druid仍能够保持100%正常运行。主要特色：为分析监控设计；快速的交互式查询；高可用；可扩展；Druid是一个开源项目，源码托管在github上。

2. [C3P0] 第三方组件

- 开源的JDBC连接池，实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。目前使用它的开源项目有Hibernate、Spring等。单线程，性能较差，适用于小型系统，代码600KB左右。

3. [DBCP]

- 官方说法BoneCP是一个高效、免费、开源的Java数据库连接池实现库。设计初衷就是为了提高数据库连接池性能，根据某些测试数据显示，BoneCP的速度是最快的，要比当时第二快速的连接池快25倍左右，完美集成到一些持久化产品如Hibernate和DataNucleus中。BoneCP特色：高度可扩展，快速；连接状态切换的回调机制；允许直接访问连接；自动化重置能力；JMX支持；懒加载能力；支持XML和属性文件配置方式；较好的Java代码组织，100%单元测试分支代码覆盖率；代码40KB左右。

4. [BoneCP]

- 官方说法BoneCP是一个高效、免费、开源的Java数据库连接池实现库。设计初衷就是为了提高数据库连接池性能，根据某些测试数据显示，BoneCP的速度是最快的，要比当时第二快速的连接池快25倍左右，完美集成到一些持久化产品如Hibernate和DataNucleus中。BoneCP特色：高度可扩展，快速；连接状态切换的回调机制；允许直接访问连

接；自动化重置能力；JMX支持；懒加载能力；支持XML和属性文件配置方式；较好的Java代码组织，100%单元测试分支代码覆盖率；代码40KB左右。

5. [Tomcat Jdbc Pool]

- Tomcat在7.0以前都是使用common-dbcp做为连接池组件，但是dbcp是单线程，为保证线程安全会锁整个连接池，性能较差，dbcp有超过60个类，也相对复杂。Tomcat从7.0开始引入了新增连接池模块叫做Tomcat jdbc pool，基于Tomcat JULI，使用Tomcat日志框架，完全兼容dbcp，通过异步方式获取连接，支持高并发应用环境，超级简单核心文件只有8个，支持JMX，支持XA Connection。

4. [Poolman]

- 太古老了

# 二、链接池的实现

## （一）实现步骤

1. 导入连接池的jar包
2. 建立连接池配置文件 `.properties` / `.xml`
3. 代码中使用

## （二）配置文件及工具类

1. Druid

- druid.properties

```
driverClassName=oracle.jdbc.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:orcl
username=scott
password=scott
# 初始连接池连接数
initialSize=5
# 最大连接池数量
maxActive=10
# 获取连接时最大等待时间，单位毫秒
maxWait=3000
# 最小连接池数量
minIdle=3
# 验证连接是否可用
validationQuery=SELECT 1 FROM dual
```

- DBUtil.java

```
/**
 * 使用Druid连接池获取链接的工具箱
 *
 * @author: liuhao
 * @version 1.0
 */
public class DBUtilDruid {
    /** 数据源 */
    private static DataSource ds;

    /** 静态代码块，在类加载时候执行 */
```

```
    static {
        Properties dbProperties = new Properties();
        try {
            dbProperties.load(DBUtilDruid.class.getResourceAsStream("/druid.properties"));
            ds = DruidDataSourceFactory.createDataSource(dbProperties);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取链接
     *
     * @return 链接对象
     */
    public static Connection getConnection() {
        Connection conn = null;
        try {
            conn = ds.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }

    /**
     * 关闭连接(不是真的关闭，还回操作)
     *
     * @param conn  链接对象
     * @param pstmt 预编译对象
     * @param rs    结果集对象
     */
    public static void close(ResultSet rs, PreparedStatement pstmt, Connection conn) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * 测试是否可以链接成功
     * @param args 主方法参数
     */
    public static void main(String[] args) {
        System.out.println(DBUtilDruid.getConnection());
```

```
    }
}
```

2. C3P0

- c3p0-config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<c3p0-config>
        <!-- 这是默认配置信息 -->
        <default-config>
                <!-- 连接四大参数配置 -->
                <property name="jdbcUrl">jdbc:oracle:thin:@localhost:1521:orcl
                </property>
                <property name="driverClass">oracle.jdbc.OracleDriver</property>
                <property name="user">scott</property>
                <property name="password">scott</property>
                <!-- 池参数配置 -->
                <property name="acquireIncrement">3</property>
                <property name="initialPoolSize">10</property>
                <property name="minPoolSize">2</property>
                <property name="maxPoolSize">10</property>
        </default-config>

        <!-- Mysql配置信息 -->
        <named-config name="oracle-config">
                <property name="jdbcUrl">jdbc:mysql://localhost:3306/mydb1</property>
                <property name="driverClass">com.mysql.jdbc.Driver</property>
                <property name="user">root</property>
                <property name="password">123</property>
                <property name="acquireIncrement">3</property>
                <property name="initialPoolSize">10</property>
                <property name="minPoolSize">2</property>
                <property name="maxPoolSize">10</property>
        </named-config>
</c3p0-config>
```

- DBUtilC3P0.java

```
/**
 * 使用C3P0连接池
 *
 * @author liuhao
 * @version 1.0
 */
public class DBUtilC3P0 {
    /** 数据源 */
    private static DataSource ds = new ComboPooledDataSource();

    /**
     * 获取数据源（连接池）
     *
     * @return 数据源对象，就是连接池
     */
    public static DataSource getDataSource() {
```

```
        return ds;
    }

    /**
     * 获取连接
     *
     * @return 数据库连接对象
     */
    public static Connection getConnection() {
        Connection conn = null;
        try {
            conn = ds.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return conn;
    }

    /**
     * 关闭连接
     *
     * @param rs    结果集
     * @param pstmt 预编译对象
     * @param conn  数据库连接
     */
    public static void close(ResultSet rs, PreparedStatement pstmt, Connection conn) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pstmt != null) {
                pstmt.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    /**
     * 测试是否可以链接成功
     * @param args 主方法参数
     */
    public static void main(String[] args) {
        System.out.println(DBUtilC3P0.getConnection());
    }
}
```

3. 自己实现

- Java连接池代码实现

```
/**
 * 自定义连接池，管理连接 代码实现：
 * 1. MyPool.java 连接池类
 * 2. 指定全局参数： 初始化数目、最大连接数、当前连接、 连接池集合
 * 3.构造函数：循环创建3个连接
 * 4. 写一个创建连接的方法
 * 5. 获取连接 ------> 判断： 池中有连接， 直接拿 ------> 池中没有连接，------> 判断，是否达到最大连接数；
 *                      达到，抛出异常；没有达到最大连接数， 创建新的连接
 * 6. 释放连接 ------> 连接放回集合中(..)
 *
 */
public class MyPool {
    private int init_count = 3; // 初始化连接数目
    private int max_count = 6; // 最大连接数
    private int current_count = 0; // 记录当前使用连接数
    // 连接池 （存放所有的初始化连接）
    private LinkedList<Connection> pool = new LinkedList<Connection>();

    // 1. 构造函数中，初始化连接放入连接池
    public MyPool() {
        // 初始化连接
        for (int i = 0; i < init_count; i++) {
            // 记录当前连接数目
            current_count++;
            // 创建原始的连接对象
            Connection con = createConnection();
            // 把连接加入连接池
            pool.addLast(con);
        }
    }

    // 2. 创建一个新的连接的方法
    private Connection createConnection() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            // 原始的目标对象
            final Connection con = DriverManager.getConnection("jdbc:mysql:///jdbc_demo", "root", "root");

            /********** 对con对象代理 **************/

            // 对con创建其代理对象
            Connection proxy = (Connection) Proxy.newProxyInstance(

                    con.getClass().getClassLoader(), // 类加载器
                    // con.getClass().getInterfaces(), // 当目标对象是一个具体的类的时候,但是这里con是一个接口
                    new Class[] { Connection.class }, // 目标对象实现的接口

                    new InvocationHandler() { // 当调用con对象方法的时候， 自动触发事务处理器
                        @Override
                        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
                            // 方法返回值
```

```
                        Object result = null;
                        // 当前执行的方法的方法名
                        String methodName = method.getName();

                        // 判断当执行了close方法的时候，把连接放入连接池
                        if ("close".equals(methodName)) {
                            System.out.println("begin:当前执行close方法开始！");
                            // 连接放入连接池
                            pool.addLast(con);
                            System.out.println("end: 当前连接已经放入连接池了！");
                        } else {
                            // 调用目标对象方法
                            result = method.invoke(con, args);
                        }
                        return result;
                    }
                });
        return proxy;
    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}

// 3. 获取连接
public Connection getConnection() {

    // 3.1 判断连接池中是否有连接，如果有连接，就直接从连接池取出
    if (pool.size() > 0) {
        return pool.removeFirst();
    }

    // 3.2 连接池中没有连接： 判断，如果没有达到最大连接数，创建；
    if (current_count < max_count) {
        // 记录当前使用的连接数
        current_count++;
        // 创建连接
        return createConnection();
    }

    // 3.3 如果当前已经达到最大连接数，抛出异常
    throw new RuntimeException("当前连接已经达到最大连接数目 ！");
}

// 4. 释放连接
public void realeaseConnection(Connection con) {
    // 4.1 判断： 池的数目如果小于初始化连接，就放入池中
    if (pool.size() < init_count) {
        pool.addLast(con);
    } else {
        try {
            // 4.2 关闭
            current_count--;
            con.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
```

```
        }
    }

    /**
     * 测试链接
     */
    public static void main(String[] args) throws SQLException {
        MyPool pool = new MyPool();
        System.out.println("当前连接: " + pool.current_count); // 3

        // 使用连接
        pool.getConnection();
        pool.getConnection();
        Connection con4 = pool.getConnection();
        Connection con3 = pool.getConnection();
        Connection con2 = pool.getConnection();
        Connection con1 = pool.getConnection();

        // 释放连接，连接放回连接池
//      pool.realeaseConnection(con1);
        /*
         * 希望：当关闭连接的时候，要把连接放入连接池！【当调用Connection接口的close方法时候，希望触发
pool.addLast(con);操作】
         * 把连接放入连接池 解决1：实现Connection接口，重写close方法 解决2：动态代理
         */
        con1.close();

        // 再获取
        pool.getConnection();

        System.out.println("连接池：" + pool.pool.size()); // 0
        System.out.println("当前连接: " + pool.current_count); // 3
    }
}
```

## （三）具体实践

- 详细请参照上传程序

1. CRUD

```
public boolean checkUser(String userName, String userPwd) {
    /** 登陆状态：true--成功，false--失败 */
    boolean result = false;
    /** 连接数据库 */
    Connection conn = null;
    /** 预编译对象 */
    PreparedStatement pstmt = null;
    /** 结果集 */
    ResultSet rs = null;
    /** 使用链接池获取连接 */
    conn = DBUtilDruid.getConnection();
    /** 定义SQL语句 */
    String checkUserSQL = "SELECT * FROM uol_user WHERE username = ? AND userpwd = ?";
```

```
    try {
        /** 准备SQL语句/创建一个执行SQL语句的对象 */
        pstmt = conn.prepareStatement(checkUserSQL);
        /** 给参数?赋值 */
        pstmt.setString(1, userName);// 给第一个参数?赋值
        pstmt.setString(2, userPwd);// 给第二个参数?赋值
        /** 执行SQL语句 */
        rs = pstmt.executeQuery();// 执行查询
        // int result = pstmt.executeUpdate();// 执行更新（添加、删除、修改）

        /** 处理执行结果 */
        if (rs.next()) {
            result = true;
        } else {
            result = false;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        // 归还连接（不是断掉）
        DBUtilDruid.close(rs, pstmt, conn);
    }
    return result;
}
```

2. 其他SQL语句的执行(DDL等)

```
String sql = "CREATE TABLE abcdef(id varchar2(10) not null,name varchar(20))";//建表语句
String sql = "DROP TABLE abcdef";//删表语句
String sql = "ALTER TABLE abcdef ADD (sex varchar2(1))";//添加类
String sql = "ALTER TABLE abcdef DROP COLUMN sex";//删除列
String sql = "ALTER TABLE abcdef MODIFY (sex number(1))";//修改列属性
String sql = "RENAME abcdef TO ab001";//修改表名
pstm.execute();//PreparedStatement,返回值boolean类型
stmt.execute(sql);//Statement
```

3. Oracle分页显示：

```
SELECT *  FROM (SELECT ROWNUM AS rowno, t.*
          FROM emp t
          WHERE   ROWNUM <= 5) table_alias
          WHERE table_alias.rowno >= 1;
```

4. 批量删除，批量添加，批量修改
5. 执行DDL语句，创建临时表，通过代码，用完删除
6. 获取元数据

- DatabaseMetaData接口:获取数据库相关信息

  > DatabaseMetaData dbms = conn.getMetaData();//创建

  - getDatabaseProductVersion
  - getDatabaseProductName
  - getDriverName

    - getDriverVersion
    - getUserName
    - getURL
- ResultSetMetaData接口:获取结果集相关信息

> ResultSetMetaData rsmd = rs.getMetaData();//创建

    - getColumnCount
    - getColumnName(i)
    - getColumnClassName(i)

7. 结果集分页：可滚动的结果集，绝对定位，conn.prepareStrtement(sql, ResultSet.xxx, ResultSet.xxx)，参数一定要写 ORM---双向同步，读（表-内存）改（内存）写（内存-磁盘）
8. 事务：ACID（属性）

- 如果设置了手动提交事务，执行完更新需要提事务

9. 配置：属性文件 Blob和Clob

- CLOB：Character Large Object，大的字符对象 常见场景：保存XML文档 方法： setAsciiStream(int parameterIndex, InputStream x, long length) getAsciiStream(String columnLabel)
- BLOB: Binary Large Object,大的二进制对象 常见场景：保存位图 方法： setBinaryStream(int parameterIndex, InputStream x, int length) getBinaryStream(String columnLabel)

10. SQL注入攻击（SQL Injection）

> 简介： 用户输入的字符串中包含了SQL指令，破坏了原SQL语句的原意，从而可以恶意登录 如userName = "1' OR '1'='1"; passWord = "1' OR '1'='1"; 防护： 用户输入时，正则验证不允许'='或'出现 接收端判断

- PreparedStatement引入：
    - 继承自Statement接口，stmt能做的pstmt都可以替代
    - SQL中未知内容使用?代替，并使用setString( , )给?设置具体的值，防止了注入的发生
- 预编译的优势：
    - 它会先将SQL语句发送给数据库进行预编译，对于相似的操作语句，他只需要预编译一次，减少编译次数
    - 安全性高，没有SQL注入风险
    - 没有SQL字符串拼接，可读性好

# 第三节：MyBatis

## 一、背景介绍

> 在2010年6月16日，iBATIS团队决定从apache迁出并迁移到Google Code，并更名为MyBatis。目前iBATIS 2.x和MyBatis 3不是100%兼容的，如配置文件的DTD变更，SqlMapClient直接由SqlSessionFactory代替了，包名也有com.ibatis变成org.ibatis等等。

1. MyBatis是一个框架，是一个第三方组件，开源的
2. 作用：用于持久化操作（数据库），是一个ORM工具。
3. 负责数据库操作的事情，MyBatis 底层依赖于JDBC
4. MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。
5. 相关网站：

- 官网：http://www.mybatis.org
- 中文开发文档：http://www.mybatis.org/mybatis-3/zh/index.html
- 中文官网（国人）：http://www.mybatis.cn/
- MyBatis源码、jar包下载：参考链接

## 二、ORM是什么

1. 全称：Object Relational Mapping 对象关系映射
2. 理解：Object--Java对象 Relational--Java和数据库 Mapping--映射
3. 类--表 对象--一条记录 属性--字段 内存--硬盘
4. 常见ORM框架：MyBatis、Hibernate、JPA

- [x] MyBatis： 半自动的ORM工具，轻量级ORM工具，需要自己写SQL语句
- [x] Hibernate（冬眠）：全自动，重量级，可以不写SQL语句，灵活性较低，性能较低
- [x] JPA：Java Persistence API的简称,是JDK 5.0注解或XML描述对象 - 关系表的映射关系，将运行期的实体对象持久化到数据库中

4. 术语：POJO、JavaBean

- [x] POJO：简单的Java对象，实际就是普通JavaBeans是。例如Users类--简单属性和简单方法（com.ychs.uol.vo.User、com.ychs.uol.model.User、com.ychs.uol.do.User）
- [x] Java Bean 命名规则(属性)：
  - 第一个单词的首字母必须小写，且单词长度大于1
  - 第二个单词首字母大写其余小写

## 三、为什么使用MyBatis?

1. 将SQL语句从Java代码中分离出，独立存放，做到解耦的效果，便于维护
2. 减少了代码量

## 四、使用MyBatis

- [x] mybatis 3.5.2 API : 参考链接

> XML:可扩展标记语言

### 实践步骤

1. 导包(MyBatis的jar包)
2. 两个配置文件

- jdbc.properties
- mybatis-config.xml

3. 数据库建表
4. 准备POJO/JavaBean---实体类
5. 写Mapper接口
6. 写配置文件：xxx.xml

- 注意链接的<类名>--使用全限定的类名（包名+类名）
- 注意链接的<方法名>还有<方法参数类型>
- <SQL语句>的书写以及<参数>的获取：#{属性名}

7. Dao实用类

- 获取工厂
- 定义接口对象（动态代理模式）
- 调用接口中的方法

8. Test测试类

- JUnit的使用
- Log4j使用

## 准备

1. xml添删改查`<insert>`、`<delete>`、`<update>`、`<select>`标签属性

- id="xxxx" ——表示此段sql执行语句的唯一标识，也是接口的方法名称【必须一致才能找到】
- parameterType="xxxx" ——表示该sql语句中需要传入的参数， 类型要与对应的接口方法的类型一致【可选】
- resultMap="xxx"—— 定义出参，调用已定义的映射管理器的id值
- resultType="xxxx"——定义出参，匹配普通Java类型或自定义的pojo【出参类型若不指定，将为语句类型默认类型，如语句返回值为int】

2. 注意： 至于为何 语句的返回值类型为什么是int，有过JDBC操作经验的朋友可能会有印象，增删改操作实际上返回的是操作的条数（受影响的条数）。而Mybatis框架本身是基于JDBC的，所以此处也沿袭这种返回值类型。

### （一）添加用户

```
<!-- 添加用户 -->
<!-- id中的insertUser就是UserMapper接口中的方法名；parameterType的值指的是insertUser方法带的参数类型，使用全限定的类名 -->
<insert id="insertUser" parameterType="com.ychs.uol.model.User">
  INSERT INTO uol_user(userid, username, userpwd, realname, sex, job,
  remark, status)
  VALUES(sys_guid(),#{userName},#{userPwd},#{realName},#{sex},#{job},#{remark},#{status})
  <!-- 特别注意：SQL语句后绝对不能加分号 -->
</insert>
```

### （二）删除用户

```
<!-- 删除用户 -->
<delete id="deleteUserByUserid" parameterType="String">
  DELETE FROM users
  WHERE userid=#{userid}
</delete>
```

### （三）修改用户

```
<!-- 修改用户 -->
<update id="modifyUser" parameterType="com.ychs.uol.model.User">
  UPDATE uol_user SET
  username = #{userName}, userpwd = #{userPwd},realname =
  #{realName},sex = #{sex},job = #{job},remark = #{remark}, status =
  #{status} WHERE userid = #{userId}
</update>
```

### （四）查询用户

- 查询所有用户

```
<!-- 查询所有用户 -->
<select id="selectAllUser" resultMap="userMap">
  SELECT * FROM uol_user
```

```
</select>

<!-- 自定义返回结果集类型，命名为：userMap，建立uol_user表字段和User类属性之间的映射关系 -->
<resultMap id="userMap" type="com.ychs.uol.model.User">
    <!-- 主键 -->
    <id property="userId" column="userid" />
    <!-- 非主键 -->
    <result property="userName" column="username" />
    <result property="userPwd" column="userpsw" />
    <result property="realName" column="realname" />
    <result property="sex" column="sex" />
    <result property="job" column="job" />
    <result property="remark" column="remark" />
    <result property="status" column="status" />
</resultMap>
```

## （五）传参（多个参数）

1. 方式一：#{0}、#{1}、#{2}

> 注意：不同版本之间使用方式不同，在3.4版本之后使用：#{arg0}、#{arg1}、#{arg2} 参考链接

- MyBatis3.4版本之前

```
<!-- 方式1: 使用序号传参 -->
<select id="selectBook" resultType="com.ychs.uol.vo.Book">
  SELECT * FROM book WHERE bookname=#{0} AND publisher=#{1}
</select>
```

- MyBatis3.4.4版本之后

```
<!-- 方式1: 使用序号传参 -->
<select id="selectBook" resultType="com.ychs.uol.vo.Book">
  SELECT * FROM book WHERE bookname=#{arg0} AND publisher=#{arg1}
</select>
```

2. 方式二：使用MyBise注解

```
//xml
  <!-- 方式2: 使用参数注解@Param -->
  <select id="selectBookByAnnotation"
    resultType="com.ychs.uol.vo.Book">
    SELECT * FROM book WHERE bookname=#{bname} AND publisher=#{pubname}
  </select>

//dao
  public Book selectBookByAnnotation(@Param("bname") String bookname, @Param("pubname")
String publisher) {
      SqlSession sqlSession =DBUtil.getSession();
      Book book = null;
      try {
        BookMapper bookMapper = sqlSession.getMapper(BookMapper.class);
        book = bookMapper.selectBookByAnnotation(bookname, publisher);
      } catch (Exception e) {
```

```
        e.printStackTrace();
      } finally {
        sqlSession.close();
      }
      return book;
  }
```

3. 方式三：使用Map(常用)

```
<!-- 方式3: 使用Map传参 -->
<select id="selectBookByMap" parameterType="java.util.Map"
  resultType="com.ychs.uol.vo.Book">
  SELECT * FROM book WHERE bookname=#{bname} AND publisher=#{pubname}
</select>
```

4. 方式四：传对象(常用)

```
<!-- 方式4: 使用JavaBean传参 -->
<select id="selectBookByBean"
  parameterType="com.ychs.uol.vo.Book" resultType="com.ychs.uol.vo.Book">
  SELECT * FROM book WHERE bookname=#{bookname} AND publisher=#{publisher}
</select>
```

## (六) 多表链接查询

- eg：专业所属院系、学校(school、department、major)

```
<resultMap id="infoMap" type="com.ychs.uol.model.Info">
  <result property="schoolName" column="schoolname" />
  <result property="deptName" column="deptname" />
  <result property="majorName" column="majorname" />
</resultMap>

<select id="seleteMajorInfo" resultMap="infoMap"
  parameterType="String">
  SELECT s.schoolname, d.deptname, m.majorname FROM
  uol_major m INNER JOIN
  uol_department d ON m.deptid = d.deptid INNER
  JOIN uol_school s ON
  d.schoolid = s.schoolid
  WHERE m.majorname=#{majorName}
</select>
```

## (七) 多条件组合查询

1. 方式一（SQL中写WHERE1=1）

```
<!-- 多条件组合查询 -->
<select id="selectLabMember" parameterType="java.util.Map" resultMap="labMemberMap">
    SELECT * FROM uol_labmember WHERE 1=1
    <if test="memberName != null">
        AND membername LIKE #{memberName}
    </if>
    <if test="grade != null">
        AND grade = #{grade}
    </if>
    <if test="school != null">
        AND school = #{school}
    </if>
</select>
```

2. 方式二（使用`<where>`标签）

```
<!-- 多条件组合查询 -->
<select id="selectLabMember" parameterType="java.util.Map" resultMap="labMemberMap">
    SELECT * FROM uol_labmember
    <where>
        <if test="memberName != null">
          AND membername LIKE #{memberName}
        </if>
        <if test="grade != null">
          AND grade = #{grade}
        </if>
        <if test="school != null">
          AND school = #{school}
        </if>
    </where>
</select>
```

## （八）分页显示

1. 方式一：使用MyBatis自带的类

```
//xml
  <!-- 分页显示，方式一：使用MyBaits自带的类 -->
  <select id="selectUserPage" resultMap="userMap">
    SELECT * FROM uol_user
  </select>

//dao
  public List<User> selectUserPage(int currPage, int pageSize) {
      List<User> userList = new ArrayList<User>();
      SqlSession sqlSession = DBUtil.getSession();
      // 偏移量，当前页第一条记录的序号（从0开始）
      int offset = (currPage - 1) * pageSize;
      try {
          UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
          userList = userMapper.selectUserPage(new RowBounds(offset, pageSize));
          sqlSession.commit();
      } catch (Exception e) {
          e.printStackTrace();
```

```
        } finally {
            DBUtil.closeSession(sqlSession);
        }
        return userList;
    }
```

2. 方式二：自己写sql(使用伪列)

```
//xml
  <!-- 分页显示，方式二：自己写SQL语句，通过参数控制 -->
  <select id="selectUserPageSql" resultMap="userMap">
    SELECT * FROM (SELECT
    ROWNUM AS rowno, u.* FROM uol_user u WHERE ROWNUM &lt;= #{arg1})
    WHERE rowno   &gt;= #{arg0}
  </select>

//dao
  public List<User> selectUserPageSql(int currPage, int pageSize) {
      List<User> userList = new ArrayList<User>();
      SqlSession sqlSession = DBUtil.getSession();
      // 当前页第一条记录的序号（从1开始）
      int begin = (currPage - 1) * pageSize + 1;
      // 当前页最后一条记录的序号
      int end = begin + (pageSize - 1);
      try {
          UserMapper userMapper = sqlSession.getMapper(UserMapper.class);
          userList = userMapper.selectUserPageSql(begin, end);
          sqlSession.commit();
      } catch (Exception e) {
          e.printStackTrace();
      } finally {
          DBUtil.closeSession(sqlSession);
      }
      return userList;
  }
```

# 五、使用MyBatis（二）

## （一）自增主键

- 参考链接
- 主键分类
  - 自然主键：没有实际意义，只是一种唯一标识
    - Oracle :使用序列
    - MySQL：自增
    - 随机数生成主键（随机数生成的种子）
    - UUID：32位的序列
  - 业务主键：带有实际意义，学号，身份证

- `<selectKey>`标签 属性|描述 |:--:|:--| |keyProperty|selectKey 语句结果应该被设置的目标属性。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。对应实体类中的主键的属性| |keyColumn|匹配属性的返回结果集中的列名称。如果希望得到多个生成的列，也可以是逗号分隔的属性名称列表。| |resultType|结果的类型。MyBatis 通常可以推算出来，但是为了更加确定写上也不会有什么问题。MyBatis 允许任何简单类型用作主键的类型，包括字符串。如果希望作用于多个生成的列，则可以使用一个包含期望属性的 Object 或一个 Map。| |order|这可以被设置为 BEFORE 或 AFTER。如果设置为 BEFORE，那么它会首先选择主键，设置 keyProperty 然后执行插入语句。如果设置为 AFTER，那么先执行插入语句，然后是 selectKey 元素| |statementType|与前面相同，MyBatis 支持 STATEMENT（直接操作sql，不进行预编译，获取数据---${xxx}），PREPARED（预处理，参数，进行预编译，获取数据---#{xxx}） 和 CALLABLE（执行存储过程） 语句的映射类型，分别代表 PreparedStatement 和 CallableStatement 类型。|
- 代码：

```
<!-- 添加用户，主键自增 -->
<insert id="insertUserKeyAutoInc"
  parameterType="com.ychs.uol.model.User">
  <selectKey keyProperty="userId" resultType="String"
    keyColumn="userNo" order="BEFORE">
    SELECT myseq.nextval as userNo FROM dual
    <!-- SELECT MAX(userid)+1 AS userNo FROM uol_user -->
  </selectKey>
  INSERT INTO uol_user(userid, username, userpwd, realname, sex, job,
  remark, status)
  VALUES(#{userId},#{userName},#{userPwd},#{realName},#{sex},#{job},#{remark},#{status})
</insert>
```

## （二）批量删除

1. `<foreach>`标签

| 属性 | 描述 |
| --- | --- |
| collection | 指定输入对象中集合属性 |
| item | 每次遍历生成的对象 |
| open | 开始遍历时拼接的串 |
| close | 结束遍历时两个对象需要拼接的串 |
| separator | 表示隔离符号 |
| index | 集合迭代位置 |

2. collection 属性详解

- 如果传入的是单参数且参数类型是一个List的时候，collection属性值为list
- 如果传入的是单参数且参数类型是一个array数组的时候，collection的属性值为array
- 如果传入的参数是多个的时候，我们就需要把它们封装成一个Map了，当然单参数也可以封装成map，实际上如果你在传入参数的时候，在breast里面也是会把它封装成一个Map的，map的key就是参数名，所以这个时候collection属性值就是传入的List或array对象在自己封装的map里面的key
- 代码：

```
<!-- 批量删除用户 -->
<delete id="deleteUserBatch" parameterType="java.util.List">
    DELETE FROM uol_user WHERE userid IN
    <foreach collection="list" item="item" open="(" separator="," close=")">
        #{item}
    </foreach>
</delete>
```

## （三）选择一个条件进行查询

- 使用`<choose>`、`<when>`、`<otherwise>`标签

```
<!-- 多条件查询用户（只能使用一个条件） -->
<select id="selectUserMulCondition"
    parameterType="java.util.Map" resultMap="userMap">
    SELECT * FROM uol_user WHERE 1 =1
    <choose>
        <when test="userName != null">
            AND username = #{userName}
        </when>
        <when test="realName != null">
            AND realname = #{realName}
        </when>
        <when test="sex != null">
            AND sex = #{sex}
        </when>
        <otherwise>
            AND sex = '女'
        </otherwise>
    </choose>
</select>
```

####（四）更新时sql动态拼装

- 使用`<set>`、`<if>`标签

```
<!-- 动态SQL: set，动态包含需要更新的列，舍去其它的 -->
<update id="updateUserIfNecessary"
    parameterType="com.ychs.uol.model.User">
    UPDATE uol_user
    <set>
        <if test="userName != null">username=#{userName},</if>
        <if test="userPwd != null">userpsw=#{userPwd},</if>
        <if test="realName != null">realname=#{realName},</if>
        <if test="sex != null">sex=#{sex},</if>
        <if test="job != null">job=#{job},</if>
        <if test="remark != null">remark=#{remark},</if>
        <if test="status != null">status=#{status}</if>
    </set>
    WHERE userid=#{userId}
</update>
```

## （四）模糊查询

1. 使用 `${}` 直接获取，不经过预编译

```
  <!-- 模糊查询，第一种写法 realname LIKE '%xxx%' 用$——不预编译 -->
  <select id="selectUserLike" parameterType="String" resultMap="userMap">
    SELECT * FROM uol_user WHERE username LIKE '%${words}%'
  </select>
```

2. 使用 `||` 链接

```
<!-- 模糊查询，第二种写法,使用Oracle字符串连接符|| -->
<select id="selectUserLike" parameterType="String" resultMap="userMap">
  SELECT * FROM uol_user WHERE username LIKE '%'||#{words}||'%'
</select>
```

3. 使用 `CONCAT()` 数据库函数链接

```
<!-- 模糊查询，第三种写法 -->
<select id="selectUserLike" parameterType="String" resultMap="userMap">
  SELECT * FROM uol_user WHERE username LIKE CONCAT(CONCAT('%',#{words}),'%')
</select>
```

## （五）级联删除

1. 调用接口中方法方式

```
public int deleteSchool(String schoolName) {
    int result = 0;
    SqlSession sqlSession = null;

    try {
        sqlSession = DBUtil.getSession();
        // 定义三个Mapper
        SchoolMapper schoolMapper = sqlSession.getMapper(SchoolMapper.class);
        DepartmentMapper departmentMapper = sqlSession.getMapper(DepartmentMapper.class);
        MajorMapper majorMapper = sqlSession.getMapper(MajorMapper.class);
        // 删除专业
        majorMapper.deleteMajor(schoolName);
        // 删除系部
        departmentMapper.deleteDepartment(schoolName);
        // 删除学校
        result = schoolMapper.deleteSchool(schoolName);
        sqlSession.commit();
    } catch (Exception e) {
        sqlSession.rollback();
        e.printStackTrace();
    } finally {
        DBUtil.closeSession(sqlSession);
    }
    return result;
}
```

2. 使用触发器

- 使用一个触发器

```
create or replace TRIGGER DELETE_SCHOOL
BEFORE DELETE ON uol_school
FOR EACH ROW
DECLARE
pragma autonomous_transaction;-- 加入一个自治事务
BEGIN
    DELETE uol_major WHERE deptid IN (SELECT deptid FROM uol_department WHERE schoolid IN
(SELECT schoolid FROM uol_school WHERE schoolname = :old.schoolname));
    DELETE uol_department WHERE schoolid IN (SELECT schoolid FROM uol_school WHERE
schoolname = :old.schoolname);
    commit;
END;
```

- 使用两个触发器

```
create or replace TRIGGER DELETE_SCHOOL
BEFORE DELETE ON uol_school
FOR EACH ROW
BEGIN
    DELETE uol_department WHERE schoolid = :old.schoolid;
END;
```

```
create or replace TRIGGER DELETE_DEPARTMENT
BEFORE DELETE ON uol_department
FOR EACH ROW
BEGIN
    DELETE uol_major WHERE deptid = :old.deptid;
END;
```

# 六、FAQ

## (一) uuid是什么

- UUID 是 通用唯一识别码（Universally Unique Identifier）的缩写
- UUID是由一组32位数的16进制数字所构成，所以UUID理论上的总数为16^32=2^128，约等于3.4 x 10^38。也就是说若每纳秒产生1兆个UUID，要花100亿年才会将所有UUID用完。
- 目前最广泛应用的UUID，是微软公司的全局唯一标识符（GUID）

## (二) xml文件里如何使用'>'，'<'？

- 参考链接
- 使用转义字符'>'---`>`，'<'---`<`

## (三) Mybatis常用的xml标签

- 参考链接

## (四) ssh和ssm的区别

- 参考链接
- ssh通常使用 Struts2为控制器(controller)， spring 为事务层(service)， hibernate 负责持久层（dao）

- ssm通常使用 springMVC为控制器(controller)，spring 为事务层(service)， MyBatis 负责持久层（dao）

# 七、MyBatis面试题

## （一）什么是MyBatis?

1. Mybatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高。
2. MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
3. 通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

## （二）Mybatis的优点

1. 把sql语句从数据库中独立出来，解除sql与程序代码之间耦合，便于代码的维护和管理。
2. 编写原生SQL语句，更加的灵活。
3. 提供XML标签（where、set、if、choose、when、otherwise、foreach、trim、bind），支持编写动态SQL语句，并可重用。
4. 封装了底层的JDBC，API调用，很好的与各种数据库兼容，并且能将结果自动的转化成JavaBean对象（结果集与java对象自动映射），简化了数据库编程的重复工作。
5. 能够与Spring很好的集成。

## （三）Mybatis的缺点

1. SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。
2. SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

## （四）MyBatis适用场合

1. MyBatis专注于SQL本身，是一个足够灵活的DAO层解决方案。
2. 对性能的要求很高，或者需求变化较多的项目，如互联网项目，MyBatis将是不错的选择。

## （五）MyBatis核心类的作用?

> 参考链接 每个基于 MyBatis 的应用都是以一个 SqlSessionFactory 的实例为中心的。SqlSessionFactory 的实例可以通过 SqlSessionFactoryBuilder 获得。而 SqlSessionFactoryBuilder 则可以从 XML 配置文件或一个预先定制的 Configuration 的实例构建出 SqlSessionFactory 的实例。

| 类名 | 作用 |
| --- | --- |
| SqlSessionFactoryBuilder | 读取全局配置文件，得到数据源信息，创建连接工厂SqlSessionFactory。它的特点是，当创建了SqlSessionFactory对象之后，这个类就不需要了。因此，它的最佳范围是存在于方法体内，也就是局部变量。 |
| SqlSessionFactory | 连接工厂；创建SqlSession实例的工厂。它的特点是，SqlSessionFactory对象一旦被创建，就无法销毁或者再创建，是单例的。因此，它存在于应用程序的整个运行生命周期 |
| SqlSession | 会话对象类，是一个接口，SqlSession 完全包含了面向数据库执行 SQL 命令所需的所有方法。可以通过 SqlSession 实例来直接执行已映射的 SQL 语句。SqlSession对应着一次数据库会话，所以每次访问数据库时都需要在SqlSessionFactory实例的openSession()方法中创建它。但一个SqlSession会话也可以执行多次SQL语句。注意：一个连接不能让多个线程同时使用,因为是非线程安全的，每个线程都应该有一个SqlSession的实例来完成对数据库的操作 |
| Mapper | 1.映射类，跟映射关系对应，是从SqlSession中获取的。<br>2.作为接口的代理类<br>3.将传入的接口类型与映射的XML文件关联起来，接口的实现是基于XML配置文件中的SQL实现，生成代理类对象<br>4.函数：session.getMapper(接口 class 实例) |

## （六）MyBatis 的工作流程或运行原理?

1. 通过SqlSessionFactoryBuilder从mybatis-config.xml配置文件中构建出SqlSessionFactory。
2. SqlSessionFactory开启一个SqlSession，通过SqlSession实例获得Mapper对象并且运行Mapper映射的Sql语句。
3. 完成数据库的CRUD操作和事务提交，关闭SqlSession。

## （七）MyBatis跟Hibernate的比较

1. MyBatis 是半自动化的,需要我们手动的编写sql。
2. Hibernate是全自动化的,只要配置映射文件,可以为我们动态的生成sql。

## （八）`#{}`和`${}`的区别是什么?

1. `${}`：简单字符串替换（属于静态文本替换），把${}直接替换成变量的值，不做任何转换，这种是取值以后再去编译SQL语句（非预编译）。一种使用情况：用于传入数据库对象，比如表名。
2. `#{}`：预编译处理，sql中的#{}替换成‘?’（占位符），补全预编译语句，有效的防止Sql语句注入，这种取值是编译好sql语句再取值。

## （九）Mybatis是如何进行分页的？分页插件的原理是什么?

1. 使用 MyBatis 提供的 RowBounds 对象进行分页。
2. 自己写分页 sql 语句实现分页。
3. 使用分页插件：分页插件的基本原理是使用Mybatis提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的sql，然后重写sql，根据dialect方言，添加对应的物理分页语句和物理分页参数。eg：`select * from student`，拦截sql后重写为：`select t.* from （select * from student）t limit 0，10`

## （十）什么是MyBatis的接口绑定,有什么好处

- 答：接口映射就是在IBatis中任意定义接口,然后把接口里面的方法和SQL语句绑定，我们直接调用接口方法就可以,这样比起原来了SqlSession提供的方法我们可以有更加灵活的选择和设置。

## （十一）接口绑定有几种实现方式,分别是怎么实现的?

1. 接口绑定有两种实现方式：

- 一种是通过注解绑定，就是在接口的方法上面加上@Select@Update等注解里面包含Sql语句来绑定。
- 另外一种就是通过xml里面写SQL来绑定，在这种情况下，要指定xml映射文件里面的namespace必须为接口的全路径名。

## （十二）什么情况下用注解绑定,什么情况下用xml绑定

答：

- 当Sql语句比较简单时候,用注解绑定。
- 当SQL语句比较复杂时候，用xml绑定，一般用xml绑定的比较多。

## （十三）MyBatis里面的动态Sql是怎么设定的?用什么语法?

答：MyBatis里面的动态Sql一般是通过if节点来实现，通过OGNL语法来实现，但是如果要写的完整，必须配合where、trim节点，where节点是判断包含节点有内容就插入where，否则不插入，trim节点是用来判断如果动态语句是以and 或or开始，那么会自动把这个and或者or取掉 。

## （十四）Xml映射文件中，除了常见的select|insert|updae|delete标签之外，还有哪些标签?

- 答：还有很多其他的标签，、、、、、，加上动态sql的9个标签，trim | where | set | foreach | if | choose | when | otherwise | bind 等，其中为sql片段标签，通过标签引入sql片段，为不支持自增的主键生成策略标签。

## （十五）Mybatis动态sql是做什么的？都有哪些动态sql？能简述一下动态sql的执行原理不?

答：

- Mybatis动态sql可以让我们在Xml映射文件内，以标签的形式编写动态sql，完成逻辑判断和动态拼接sql的功能，Mybatis提供了9种动态sql标签trim|where|set|foreach|if|choose|when|otherwise|bind。
- 其执行原理为，使用OGNL从sql参数对象中计算表达式的值，根据表达式的值动态拼接sql，以此来完成动态sql的功能。

## （十六）Mybatis是如何将sql执行结果封装为目标对象并返回的？都有哪些映射形式?

答：

- 第一种是使用标签，逐一定义列名和对象属性名之间的映射关系。
- 第二种是使用sql列的别名功能，将列别名书写为对象属性名，比如T_NAME AS NAME，对象属性名一般是name，小写，但是列名不区分大小写，Mybatis会忽略列名大小写，智能找到与之对应对象属性名，你甚至可以写成T_NAME AS NaMe，Mybatis一样可以正常工作。有了列名与属性名的映射关系后，Mybatis通过**反射**创建对象，同时使用反射给对象的属性逐一赋值并返回，那些找不到映射关系的属性，是无法完成赋值的。

## （十七）Mybatis的Xml映射文件中，不同的Xml映射文件，id是否可以重复?

答：

- 不同的Xml映射文件，如果配置了namespace，那么id可以重复；如果没有配置namespace，那么id不能重复；毕竟namespace不是必须的，只是最佳实践而已。
- 原因就是namespace+id是作为Map<String, MappedStatement>的key使用的，如果没有namespace，就剩下id，那么，id重复会导致数据互相覆盖。有了namespace，自然id就可以重复，namespace不同，namespace+id自然也就不同。

### （十八）Mybatis映射文件中，如果A标签通过include引用了B标签的内容，请问，B标签能否定义在A标签的后面，还是说必须定义在A标签的前面?

答：虽然Mybatis解析Xml映射文件是按照顺序解析的，但是，被引用的B标签依然可以定义在任何地方，Mybatis都可以正确识别。 原理是，Mybatis解析A标签，发现A标签引用了B标签，但是B标签尚未解析到，尚不存在，此时，Mybatis会将A标签标记为未解析状态，然后继续解析余下的标签，包含B标签，待所有标签解析完毕，Mybatis会重新解析那些被标记为未解析的标签，此时再解析A标签时，B标签已经存在，A标签也就可以正常解析完成了。

3. ORM 思想是什么，有什么好处?
    - 将数据库代码和Java代码分离，实现解耦，便于维护
    - 参考文章
4. 常见的ORM框架有哪些?
    - MyBatis：流行，在SSM中使用
    - Hibernate：sql固定生成，不够灵活。
    - JPA：java持久化API
    - SSH和SSM区别
5. MyBatis中使用到哪些设计模式?

- 工厂模式
- 代理模式（动态代理）

两种调用sql方式 1.直接通过sqlSesson调用添删改查方法，带有侵入性，不能面向接口编程

2. 使用接口来进行
3. 使用注解：没有xml，零配置，但是改动需要编译

## 八、进阶面试题

### （一）MyBatis比IBatis比较大的几个改进是什么

1. 有接口绑定,包括注解绑定sql和xml绑定Sql ,
2. 动态sql由原来的节点配置变成OGNL表达式,
3. 在一对一，一对多的时候引进了association,在一对多的时候引入了collection节点,不过都是在resultMap里面配置

### （二）MyBatis实现一对一有几种方式?具体怎么操作的

答：有联合查询和嵌套查询，分别如下

- 联合查询是几个表联合查询,只查询一次，通过在resultMap里面配置association节点配置一对一的类就可以完成。
- 嵌套查询是先查一个表,根据这个表里面的结果的外键id,去再另外一个表里面查询数据,也是通过association配置,但另外一个表的查询通过select属性配置。

### （三）IBatis和MyBatis在核心处理类分别叫什么

答：

- IBatis里面的核心处理类交SqlMapClient
- MyBatis里面的核心处理类叫做SqlSession

### （四）讲下MyBatis的缓存

答：MyBatis的缓存分为一级缓存和二级缓存，分别如下：

- 一级缓存放在session里面，默认就有。

- 二级缓存放在它的命名空间里，默认是打开的。使用二级缓存属性类需要实现Serializable序列化接口(可用来保存对象的状态),可在它的映射文件中配置

### （五）最佳实践中，通常一个Xml映射文件，都会写一个Dao接口与之对应，请问，这个Dao接口的工作原理是什么？Dao接口里的方法，参数不同时，方法能重载吗？

答：

- Dao接口，就是人们常说的 Mapper 接口，接口的全限名，就是映射文件中的namespace的值，接口的方法名，就是映射文件中 MappedStatement 的id值，接口方法内的参数，就是传递给sql的参数。Mapper接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为key值，可唯一定位一个MappedStatement，举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到namespace为com.mybatis3.mappers.StudentDao下面id = findStudentById的MappedStatement。在Mybatis中，每一个`<select>、<insert>、<update>、<delete>`标签，都会被解析为一个MappedStatement对象。
- Dao接口里的方法，是不能重载的，因为是全限名+方法名的保存和寻找策略。
- Dao接口的工作原理是JDK动态代理，Mybatis运行时会使用JDK动态代理为Dao接口生成代理proxy对象，代理对象proxy会拦截接口方法，转而执行MappedStatement所代表的sql，然后将sql执行结果返回。

### （六）简述Mybatis的插件运行原理，以及如何编写一个插件。

答：

- Mybatis仅可以编写针对ParameterHandler、ResultSetHandler、StatementHandler、Executor这4种接口的插件，Mybatis使用JDK的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这4种接口对象的方法时，就会进入拦截方法，具体就是InvocationHandler的invoke()方法，当然，只会拦截那些你指定需要拦截的方法。
- 实现Mybatis的Interceptor接口并复写intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。

### （七）Mybatis执行批量插入，能返回数据库主键列表吗？

- 答：能，JDBC都能，Mybatis当然也能。

### （八）Mybatis是否支持延迟加载？如果支持，它的实现原理是什么？

答：

- Mybatis仅支持association关联对象和collection关联集合对象的延迟加载，association指的就是一对一，collection指的就是一对多查询。在Mybatis配置文件中，可以配置是否启用延迟加载lazyLoadingEnabled=true|false。
- 它的原理是，使用CGLIB创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用a.getB().getName()，拦截器invoke()方法发现a.getB()是null值，那么就会单独发送事先保存好的查询关联B对象的sql，把B查询上来，然后调用a.setB(b)，于是a的对象b属性就有值了，接着完成a.getB().getName()方法的调用。这就是延迟加载的基本原理。

### （九）Mybatis中如何执行批处理？

- 答：使用BatchExecutor完成批处理。

### （十）Mybatis都有哪些Executor执行器？它们之间的区别是什么？

答：

- Mybatis有三种基本的Executor执行器，SimpleExecutor、ReuseExecutor、BatchExecutor。
- SimpleExecutor：每执行一次update或select，就开启一个Statement对象，用完立刻关闭Statement对象。

- ReuseExecutor：执行update或select，以sql作为key查找Statement对象，存在就使用，不存在就创建，用完后，不关闭Statement对象，而是放置于Map<String, Statement>内，供下一次使用。简言之，就是重复使用Statement对象。
- BatchExecutor：执行update（没有select，JDBC批处理不支持select），将所有sql都添加到批处理中（addBatch()），等待统一执行（executeBatch()），它缓存了多个Statement对象，每个Statement对象都是addBatch()完毕后，等待逐一执行executeBatch()批处理。与JDBC批处理相同。
- 作用范围：Executor的这些特点，都严格限制在SqlSession生命周期范围内。

### （十一）Mybatis中如何指定使用哪一种Executor执行器?

答：

- 在Mybatis配置文件中，可以指定默认的ExecutorType执行器类型，也可以手动给DefaultSqlSessionFactory的创建SqlSession的方法传递ExecutorType类型参数。

### （十二）Mybatis是否可以映射Enum枚举类?

答：

- Mybatis可以映射枚举类，不单可以映射枚举类，Mybatis可以映射任何对象到表的一列上。映射方式为自定义一个TypeHandler，实现TypeHandler的setParameter()和getResult()接口方法。TypeHandler有两个作用，一是完成从javaType至jdbcType的转换，二是完成jdbcType至javaType的转换，体现为setParameter()和getResult()两个方法，分别代表设置sql问号占位符参数和获取列查询结果。

### （十三）简述Mybatis的Xml映射文件和Mybatis内部数据结构之间的映射关系?

- 答：Mybatis将所有Xml配置信息都封装到All-In-One重量级对象Configuration内部。在Xml映射文件中，`<parameterMap>`标签会被解析为ParameterMap对象，其每个子元素会被解析为ParameterMapping对象。`<resultMap>`标签会被解析为ResultMap对象，其每个子元素会被解析为ResultMapping对象。每一个`<select>`、`<insert>`、`<update>`、`<delete>`标签均会被解析为MappedStatement对象，标签内的sql会被解析为BoundSql对象。

# 第五章 JavaWeb

## 第一节：Web应用服务器

### 一、什么是Web应用服务器

- 随着Internet的发展壮大,“主机/终端”或“客户机/服务器”的传统的应用系统模式已经不能适应新的环境,于是就产生了新的分布式应用系统,相应地,新的开发模式也应运而生，即所谓的“浏览器/服务器”结构、“瘦客户机”模式。应用服务器便是一种实现这种模式核心技术。
- Web应用程序驻留在应用服务器(Application Server)上。应用服务器为Web应用程序提供一种简单的和可管理的对系统资源的访问机制。它也提供低级的服务，如HTTP协议的实现和数据库连接管理。Servlet容器仅仅是应用服务器的一部分。除了Servlet容器外，应用服务器还可能提供其他的Java EE(Enterprise Edition)组件，如EJB容器，JNDI服务器以及JMS服务器等。

### 二、常见的Web应用服务器

- Tomcat：来自于Apache组织
- Weblogic：美国的BEA,被Oracle收购，收费
- Websphere：来自IBM,付费的

- JBoss：RedHat（生产Linux厂家之一） 开源免费

## 三、Tomcat

> Apache组织 Apache软件基金会（也就是Apache Software Foundation，简称为ASF），是专门为运作一个开源软件项目的 Apache 的团体提供支持的非盈利性组织，这个开源软件项目就是 Apache 项目。这个组织把自己作为有着相同目标的开发者与用户的团体，而不是简单的共享在一个服务器上的一组项目的组织团体。在它所支持的 Apache 项目与子项目中，所发行的软件产品都遵循 Apache许可证（Apache License）。 Tomcat Maven XML

### （一）背景介绍

1.
2. 官网：tomcat.apache.org（来自于apache组织，开源的）
3. 是什么?

- 是一个Web**应用**服务器： 参考链接
- 也是一个Servlet的容器,服务于Servlet(服务器端的小程序)

3. 作用：
    - 部署我们开发好的Web项目，用户通过浏览器访问我们的项目。
    - 把JSP网页编译成HTML,便于浏览器进行显示
4. 调试输出包：com-loging,一种日志信息包，提示信息都为红色

### （二）Tomcat目录结构：

- apache-tomcat-8.5.43\conf\server.xml中找到8080，更改端口号
- conf：用于配置Tomcat
- lib：Tomcat服务器的jar包
- webapps：放置部署到服务器的项目，Web应用程序
- work目录：JSPA网页被Tomcat编译成.java文件，该目录下放置的就是编出出来的JSP网页。 ####（三）web服务器和应用服务器的区别 参考链接

### （四）FAQ

- 端口被占用：
    - eclipse中，双击tomcat，编辑HTTP端口号，四位数字
- 提示超时（time outs）
    - eclipse中，双击tomcat，编辑Timeouts，增大时间

## 四、面试题

1. 什么是 Servlet Container 参考链接
2. Servlet Container的作用是什么?

- web应用服务器
- 接收用户/浏览器的请求，执行JSP网页

# 第二节：JSP

## 一、背景介绍

- 动态网页(JSP,ASP,PHP)
- 静态网页（HTM,HTML）

- JSP：JAVA Server Pages---Java服务端页面
- JSP、Servlet属于Java EE范畴，是Jave EE标准技术（ORACLE维护）
- 主要作用：View层，主要通过图形化的界面显示给用户与用户交互
- `<% %>`：JSP的基本语法，Page指令
- 可以导包，但不推荐使用`<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" imports="java.util.*"%>`
- JSP网页浏览器是不能直接执行的，必须由Tomcat编译成.java文件，然后将一些信息发挥浏览器，Tomcat发给浏览器的内容就是：CSS、JS、HTML
- JSP的动态包含和静态包含
- Jsp的执行过程
  - jsp---java---class---html
  - jsp的本质就是一个Servlet
- JSP语法

```
<%  %> //放Java代码
<%= %> //输出变量值
<%! %> //在网页上声明变量
```

- JSP文件结构

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
</head>
<body>

</body>
</html>
```

# 二、JSP的内置对象

> （直接用，不用创建）【9种】

1. page：代表当前网页，声明周期就是当前页
2. request：请求
3. response：响应
4. sesson---HttpSesson：会话,一个上网者一个，通过sessonid标识
5. application---ServletContext：服务器启动到服务器关闭
6. out：out.print()输出
7. config：配置
8. pageContext：页面上下文，基本不用
9. exception：异常

# 三、使用的技术

## （一）JSTL、EL

> 用标签替代网页上的Java代码 https://www.runoob.com/jsp/jsp-jstl.html

1. 导包

- jstl.jar
- standard.jar

2. 使用

- `${}`用来取Servlet转发回到数据
- `<c:forEach>`、`<c:if>` 用法参考
- JSTL可能出现的异常：参考链接

## （二）JSON

> JSON(JavaScript Object Notation, JS 对象简谱)

- 基本使用语法

```
var user = {
  'userid' :100,
  'userName': 'zhangsan',
  'sex': '男',
  'address':{
      'homeAddress':'大同',
      'schoolAddress':'太原'
  }
};
// 两种方式
user.userName;
user['userName'];

user.address.homeAddress;

var userArray = {
    {
      'userid' :100,
      'userName': 'zhangsan',
      'sex': '男'
    },
    {
      'userid' :100,
      'userName': 'zhangsan',
      'sex': '男'
    }
}
userArray[1].userName;
```

## （三）AJAX

1. Ajax概念

- Ajax全称为“Asynchronous JavaScript and XML”，异步的JavaScript和XML
- Ajax并不是一门新的语言或技术，它实际上是几项技术按一定的方式组合在一起，在共同的协作中发挥各自的作用。

2. Ajax所使用的技术包括：

- 技术：JavaScript、XML、CSS、XMLHttpRequest、JSON
- 异步：发送请求以后，不等结果，由回调函数处理。
- JavaScript:向服务器发送请求，获得返回结果，更新页面
- XML：用来封装数据
- XMLHttpRequest：进行异步数据读取

3. 使用Ajax优势

- 不需要安装插件支持
- 局部刷新、提高用户的体验度，数据从服务器商加载
- 减轻服务器和带宽的负担

4. 核心对象：XMLHttpRequest

- 通过该对象向服务器发送请求。
- 它是异步请求（无需刷新页面便可向服务器传输或读写数据）的技术，所有现代浏览器都支持（Chrome、IE5+）

7. 如何创建XMLHttpRequest对象

- IE7.0以下是以ActiveXObject的方式引入XMLHttpRequest对象的，创建方式如下：

```
//IE5.0 IE6.0
xmlHttpReq = new ActiveXObject("Microsoft.XMLHTTP");
```

- 其它浏览器比如 IE7.0+，Firefox、chrome等都支持原生的XMLHttpRequest对象，创建方式如下：

```
xmlHttpReq = new XMLHttpRequest();
```

- 通用的创建方式

```
var xmlHttp;//声明一个保存XMLHttpRequest的变量
function createXHR(){
        if(windows.ActiveObject){
                xmlHttp = new ActiveObject("Microsoft XMLHTTP");//IE5,IE6
        }else{
        xmlHttp = new XMLHttpRequest();//IE7.0+ 以及其它浏览器
        }
}
```

8. XMLHttpRequest对象的属性与方法

- 方法

```
// method参数：设置请求类型，主要有get和post请求
// url参数：请求地址，可以是相对地址，也可以是绝对地址
// asynchronous参数：默认true为异步请求，false为同步请求
xhr.open(method,url,asynchronous);用来向服务器建立连接

// 参数：提交的内容。
// POST方式:data就是提交的参数，send(username=root&password=abc123);
// GET方式:send(null)
send(data):发送请求
```

- 属性

```
// onreadystatechange:设置状态改变时的回调函数，回调函数用来获取服务器数据。
onreadystatechange=function(){
  // TODO
}

// 在收到服务器端响应后，响应的数据会自动填充到XHR对象的属性中
// 当请求完成加载（readyState值为4）并且响应已经成功（status值为200）时，就可以处理服务端的返回结果了。
[readyState]:服务器状态响应
  状态码：
    0：未初始化
    1：正在加载
    2：加载完成
    3：请求进行中
    4：请求完成

[responseText]:服务器返回的数据(文本格式)

[responseXML]:服务器返回的数据（XML格式）
```

9. 使用XMLHttpRequest的步骤

```
1)创建XMLHttpRequest对象

2）设置请求的方法及URL
xhr.open("GET/POST","url",true/false); //true表示异步请求，false表示同步请求

3）设置状态改变时的回调函数
       xhr.onreadystatechange=function(){
         // TODO
       }
       0:未初始化
       1:正在加载
       2:加载完成
       3：请求进行中
       4：请求完成
4）发送请求
       xhr.send(data),
       如果为post提交，则data为提交的数据，如果为get提交，则参数为null即可。
```

## 四、CSS前端框架模板

- 饿了吗Element
- MUI
- LayUI
- jQuery
- Easy UI
- 妹子 UI
- H UI

# 第三节：Servlet

## 一、背景介绍

1. 服务器端小程序（service + let（传单））
2. 使用java编写
3. 依赖于Tomcat、和JDK
4. 充当控制层（Controller）,Jsp网页和Service之间的桥梁，处理用户请求。

```
JSP(HTML-vue)---Servlet(Spring)---Service(Spring MVC)---Dao(MyBatis)---DB
```

5. Servlet和JSP有何区别和联系

- 什么是servlet？ （1）Servlet是一种服务器端的Java应用程序，具有独立于平台和协议的特性，可以生成动态的Web页面。 （2）它担当客户请求（Web浏览器或其他HTTP客户程序）与服务器响应（HTTP服务器上的数据库或应用程序）的中间层。
- 什么是jsp？ JSP全名为：Java Server Pages，中文名叫Java服务器页面，其根本是一个简化的Servlet设计，由Microsystems公司倡导，许多公司参与一起建立的一种动态网页技术标准。
- servlet和jsp的联系？
  - jsp是对servlet的一种高级封装，本质还是servlet。
  - 第一次加载jsp页面的时候，会生成一个java文件，虚拟机编译为.class文件，最后加载并初始化为一个servlet
- servlet和jsp的共同点和不同点： （1）servlet在Java代码中通过HttpServletResponse来动态生成一个html页面 （2）jsp是通过把java代码嵌入到html中去生成一个动态的html页面 （一个是在java中写html，另一个是在html中写java代码）
- jsp和servlet出现的意义： （1）当我们使用servlet来生成动态页面的时候，会非常的复杂。因为需要在out.println()来写入html语句。但是servlet在处理前端和后台数据交互的时候有特别的优秀。 （2）因此在这个时候我们引入jsp技术来替代servlet生成html的功能。让serlvet只专注前端页面和后台数据的交互。从而也是实现来mvc的思想。

# 二、使用Servlet

####（一）创建Servlet类：

- 继承HttpServlet类
- 重写两个方法：doGet()、doPost()---浏览器向服务器发送的请求常用的两种：get、post
- 使用注解配置Servlet的请求路径,JSP表单的action调用
- Servletd的配置也可以在web.xml里(Servlet3.0版本之后新增了Servlet注解的功能)
  `@WebServlet("/AddCourseMemberServlet")`
- get：
  - 在浏览器的地址栏，输入地址，敲回车，属于get方式；
  - 超链接发出的请求也属于get方式
  - 表单method是get
- post：
  - 表单method是post

####（二）查看API

1. HttpServletRequest:

- 获得客户机信息

```
String requestUrl = request.getRequestURL().toString();// 得到请求的URL地址
String requestUri = request.getRequestURI();// 得到请求的资源
String queryString = request.getQueryString();// 得到请求的URL地址中附带的参数
String remoteAddr = request.getRemoteAddr();// 得到来访者的IP地址
String remoteHost = request.getRemoteHost();
int remotePort = request.getRemotePort();
String remoteUser = request.getRemoteUser();
String method = request.getMethod();// 得到请求URL地址时使用的方法
String pathInfo = request.getPathInfo();
String localAddr = request.getLocalAddr();// 获取WEB服务器的IP地址
String localName = request.getLocalName();// 获取WEB服务器的主机名
```

- 获取查询字符串

```
request.setAttribute();
request.getAttribute();
request.removeAttribute();
```

- 转发

```
request.getRequestDispatcher("publicclass/selectClass.jsp").forward(request, response);
```

- Session

```
HttpSession session = request.getSession();// 获取Sesson
```

- Cookies

```
Cookie[] cookies = request.getCookies();
```

2. HttpServletResponse

- 重定向

```
response.sendRedirect("login.jsp");// 重定向页面
```

- 设置编码

```
response.setContentType("text/html;charset=utf-8");
response.setCharacterEncoding("UTF-8");
PrintWriter out = response.getWriter();
```

3. HttpSession

- 设置最大的生命周期时间（单位/s）

```
setMaxInactiveInterval()
```

- 让sesson失效

```
session.invalidate();
```

- 删除指定session

```
session.removeAttribute("userName");
```

4. 关于重定向和转发

- 重定向：`response.send("login.jsp")`
  - 地址栏发生变化
  - 浏览器服务器交互两次，request对象无法共享
  - 执行重定向代码后的代码还会继续执行
- 转发：`request.getRequestDispatcher("login.jsp").forward(request, response);`
  - 地址栏不发生变化就能跳转，服务器内部完成（内部资源的跳转，共享request对象），转发过程浏览器不知道
  - request对象的生命周期：请求响应结束对象就销毁
  - 执行转发代码后的代码不会被继续执行

# 三、使用的技术

## （一）过滤器：Filter

> 参考：https://blog.csdn.net/yuzhiqiang_1993/article/details/81288912

```
// new一个过滤器（三个参数：请求，响应，链条）
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
```

- Jsp和Servlet之间的，用于校验信息、乱码处理（用过滤器filter设置编码格式）
- 请求、相应乱码原因：https://www.jb51.net/article/94408.htm

2.拦截器与Filter的区别

- Spring的拦截器与Servlet的Filter有相似之处，比如二者都是AOP编程思想的体现，都能实现权限检查、日志记录等。
- 不同的是：
  - 使用范围不同：Filter是Servlet规范规定的，只能用于Web程序中。而拦截器既可以用于Web程序，也可以用于Application、Swing程序中。
  - 规范不同：Filter是在Servlet规范中定义的，是Servlet容器支持的。而拦截器是在Spring容器内的，是Spring框架支持的。
  - 使用的资源不同：同其他的代码块一样，拦截器也是一个Spring的组件，归Spring管理，配置在Spring文件中，因此能使用Spring里的任何资源、对象，例如Service对象、数据源、事务管理等，通过IoC注入到拦截器即可；而Filter则不能。

3. 过滤器的执行顺序和url-pattern标签匹配的精确程度（/*，/xxxServlet）无关，只和他们的filter-mapping标签在web.xml文件中的顺序有关,靠上的配置的先执行。

## （二）监听器：Listener

1. 监听：request的创建、销毁，以及属性变化
2. 启动：服务器启动就同步启动

## （三）Session（会话）

- 每个用户有一个Session
- 有唯一的Jsessionid表示
- 使用Session可以记录单个用户的信息（eg:username）

```
HttpSession session = request.getSession();// 获取Sesson
session.setAttribute("username", userName);
```

## (四) Cookies

1. 把一些后台信息存储到浏览器端

- 放到缓存里
- 放到硬盘上

2. 具体实现

- 后端：https://blog.csdn.net/w_linux/article/details/79769256
- 前端：https://blog.csdn.net/qq_36834445/article/details/79922821
- cookies详解：https://www.cnblogs.com/wanghuaying/p/9557210.html
- Cookie 的路径以及 Cookie 域：https://blog.csdn.net/czh500/article/details/80220637

3. 使用cookies记住密码

```
<!-- 从cookies获取用户名以及用户密码 -->
<%
    // 获取Cookies中的密码用于记住密码
    String username = ""; //用户名
    String userpwd = ""; //密码

    Cookie[] cookies = request.getCookies();
    //保存有cookie对象
    if(cookies != null && cookies.length > 0){
        for(Cookie c: cookies){
            if(c.getName().equals("userName")){
                username = c.getValue();
                break;
            }
            if(c.getName().equals("userPwd")){
                userpwd = c.getValue();
            }
        }
    }
%>

<!-- 将获取到的用户名以及用户密码填入 -->
<li><input name="username"type="text" class="loginuser" style="color: gray" value="
<%=username%>" placeholder="请输入用户名" /></li>
            <li><input name="userpwd" id="userpwd" type="password" class="loginpwd"
style="color: gray" value="<%=userpwd%>" placeholder="请输入密码" /></li>
```

4. 使用Ajax实现自动登陆

```
<!-- 实现自动登陆 -->
<script type="text/javascript">
    // 获取指定cookie的值
    function getCookie(cname){
        var name = cname + "=";
        var ca = document.cookie.split(';');
        for(var i=0; i<ca.length; i++) {
            var c = ca[i].trim();
            if (c.indexOf(name)==0) {
                return c.substring(name.length,c.length);
            }
        }
        return "";
    }

    // 获得cookie中存放的密码实现自动登陆
    function checkCookie()
    {
        var username=getCookie("username");
        var userpwd=getCookie("userpwd");
        $("#username").val(username);
        $("#userpwd").val(userpwd);
        // alert("username: "+ username);
        // alert("userpwd: "+ userpwd);

        if (username!="" && userpwd!="") {
            //location.href="AutoLoginServlet?username="+username+"&userpwd="+userpwd;
        } else {
          //alert("Please login !");
        }
    }
</script>
```

### （四）web.xml加载次序

- Listener----Filter----Servlet

# 第四节：UOL联合开放实验室开发项目开发

## 一、技术点

1. Tomcat、jdk、Servlet的版本对应关系
2. Seclect向网页传递参数的一种方式 将集合保存到request, 跳转到指定网页（转发），链式编程
3. 超链接传参数(属于get传参方式)

```
<a href="DeleteUserServlet?userid=${user.userid}&userpwd=${user.userpwd}"/>
```

4. chrom调试

- XHR
- Console、NetWork、
- HTTP协议/头部/请求/响应---无状态协议无法跟踪用户

- sesson:一个用户产生一个sessonid，sessonid在服务器上保留一份，在cookis中也保存一份，在浏览器向服务器发送请求的时候，会发送cookie中的sessonid，服务器进行比对，相同也是一个人，不同则不是同一个人，浏览器关闭
- 查询Session原理
- 查询字符串：QueryString

# 二、功能模块实现

## （一）MD5加密

- MD5加密类

```
package com.ychs.uol.util;
/**
* @项目名称: EncryptionCode
* @文件名称: MyMD5Util.java
* @Date: 2019年7月28日
* @Copyright: 2019 www.xxx.com Inc. All rights reserved.
* 注意：本内容仅限于xxx公司内部传阅，禁止外泄以及用于其他的商业目的
*/

import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/**
 * 简单的加密工具类
 * 参考：https://blog.csdn.net/flower_CSDN/article/details/80058684
 *
 * @author: liuhao
 * @version 1.0
 */
public class MyMD5Util {

    /**
     * 对原始密码进行加密
     *
     * @param password 原始密码
     * @return 使用MD5加密后的密码
     */
    public static String getEncryptedPwd(String password) {
        String newPassword = "";
        try {
            // 生成一个MD5加密计算摘要
            MessageDigest md = MessageDigest.getInstance("MD5");
            // 计算md5函数
            md.update(password.getBytes());
            // digest()最后确定返回md5 hash值，返回值为8为字符串。因为md5 hash值是16位的hex值，实际上就是8位的字符
            // BigInteger函数则将8位的字符串转换成16位hex值，用字符串来表示；得到字符串形式的hash值
            newPassword = new BigInteger(1, md.digest()).toString(16);
        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
        }
        return newPassword;
    }
```

```
    /**
     * 验证密码是否正确
     *
     * @param password     当前获取的密码
     * @param passwordInDb 数据库查到的密码
     * @return 验证成功返回true，失败返回false
     */
    public static boolean validatePassword(String password, String passwordInDb) {
        boolean result = false;
        if (getEncryptedPwd(password).equals(passwordInDb)) {
            result = true;
        } else {
            result = false;
        }
        return result;
    }

    /**
     * @param args 加密测试
     *
     */
    public static void main(String[] args) {
        // 解密网站：https://www.cmd5.com/

        // 定义原始密码
        String oldStr = "liuhao";
        System.out.println("加密前：" + oldStr);

        // 使用MD5加密后的密码
        String newStr = getEncryptedPwd(oldStr);
        System.out.println("加密后：" + newStr);

        // 验证密码时候正确
        boolean result = validatePassword(oldStr, newStr);
        System.out.println("验证密码结果：" + result);
    }
}
```

## （二）实现二级联动

1. JSON

- 前端使用js解析
  - jQuery：Ajax函数，$.getJSON{}
- 后端使用FastJSON（阿里）

2. 具体实现

- 前端程序

```
<script type="text/javascript">
   function getDept() {
       clearDept();
       var param = {
```

```
            'schoolId' : $("#school").val(),
            'ram' : Math.random()
        };
        $.getJSON("FindDeptBySchoolServlet", param, function(data) {
            $.each(data, function(index, item) {
                var op = $("<option>").val(item.deptId).text(item.deptName);
                $("#department").append(op);
            });
        });
    }
    function getMajor() {
        clearMajor();
        var param = {
            'schoolId' : $("#school").val(),
            'deptId' : $("#department").val(),
            'ram' : Math.random()
        };
        $.getJSON("FindMajorByDeptServlet", param, function(data) {
            $.each(data, function(index, item) {
                var op = $("<option>").val(item.majorId).text(item.majorName);
                $("#major").append(op);
            });
        });
    }
    function clearDept() {
        $("#department").html("");
        var pleaseOption = $("<option>").val("").text("请选择").prop('selected',true);
        $("#department").append(pleaseOption);
    }
    function clearMajor() {
        $("#major").html("");
        var pleaseOption = $("<option>").val("").text("请选择").prop('selected',true);
        $("#major").append(pleaseOption);
    }
</script>
```

- 后端Servlet

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    response.setContentType("text/html;charset=utf-8");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();

    SchoolService schoolService = new SchoolServiceImpl();
    List<School> schoolList = schoolService.selectAllSchool();
    // 使用阿里的插件，转为JSON字符串
    String jsonStr = JSON.toJSONString(schoolList);
    out.print(jsonStr);
    out.close();
}
```

## （三）用户名是否被占用

1. JSON

2. FastJSON 第三方组件，阿里开源的---fastjson-1.2.7.jar
3. JSON 和 Java对象的转换使用FastJSON
4. 不止有着一种，谷歌也有
5. JQuery：AJAX
6. 具体实现

- 前端相关程序

```
<script type="text/javascript">
    //定义JSON对象，获得用户输入框的值
    function checkUserName() {
        var param = {
                'userName':$("#username").val()
        };

        // 通过JSON将得到的用户名传到Servlet,检测并返回结果
        $.getJSON("CheckUserNameServlet",param,function(data){
            // 处理服务器发来的数据，将返回的数据显示到输入框后，进行提示
            $("#usernameTip").text(data);
        });
    }
</script>
```

- 后端Servlet程序

```
protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
    UserService userService = new UserServiceImpl();
    response.setContentType("text/html;chartset=utf-8");
    response.setCharacterEncoding("UTF-8");
    PrintWriter out = response.getWriter();

    String userName = request.getParameter("userName");
    boolean result = userService.checkUserName(userName);
    String hint = "";
    if(result) {
        hint = "用户被占用! ";
    }
    String jsonStr = JSON.toJSONString(hint);
    out.print(jsonStr);
}
```

## （四）分页

1. 通过Servlet
2. 通过Ajax

- 前端程序

```
<!-- 实现分页 -->
<script  type="text/javascript">
   function findPublicClass(page){
           // 清除原来的内容
           clearTbaby();
```

```
            // 准备查询的参数
            var param = {
                'page':page,
              'courseName':$("#courseName").val(),
              'startDate':$("#startDate").val(),
              'endDate':$("#endDate").val(),
              'school':$("#school").val(),
              'department':$("#department").val(),
              'grade':$("#grade").val(),
                'ram':Math.random()
            };
            // console.log(param);
            // 调用Servlet进行查询,并将查询结果进行拼装
            $.getJSON("SelectOpenCourseAjaxServlet?status=second", param, function(data) {
                    $.each(data,function(index,item){
                            var str = $("<tr>").append($("<td>").text(item.courseDate))
                                               .append($("<td>").text(item.courseName))
                                    .append($("<td>").text(item.speakerName))
                                    .append($("<td>").text(item.organizerName))
                                    .append($("<td>").text(item.place))
                                    .append($("<td>").append($("<a>").text("显示人员")));
                            $("#tbaby").append(str);
                    });
            });

            // 首页
            var firstPage = $("<a href='javascript:void(0)'>首页</a>").click(function(){
                    findPublicClass(1);
            });
            // 上一页
            var prePage = $("<a href='javascript:void(0)'>上一页</a>").click(function(){
              findPublicClass(page - 1);
          });
            // 下一页
            var nextPage = $("<a href='javascript:void(0)'>下一页</a>").click(function(){
              findPublicClass(page + 1);
          });
            // 尾页
            var endPage = $("<a href='javascript:void(0)'>尾页</a>").click(function(){
              findPublicClass(${pageNum});
          });
        // 提示标签
        var span = $("<span>").text("当前第"+ page + "页 共" + ${pageNum} + "页");
```

```
        $("#pagediv").html("");
```

if(page == 1){ $("#pagediv").append(nextPage).append(endPage).append(span); } if(page != 1 && page != ${pageNum}){ $("#pagediv").append(firstPage).append(prePage).append(nextPage).append(endPage).append(span); } if(page == ${pageNum}){ $("#pagediv").append(firstPage).append(prePage).append(span); } }

```
function clearTbaby(){
```

$("#tbaby").html(""); }

- 后端程序
  ```java
  protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {
  OpenCourseService openCourseService = new OpenCourseServiceImpl();

  String status = EncodingUtil.changeEncoding(request.getParameter("status"));
  if (status.equals("first")) {
     // 查询到的所有学校信息
     SchoolService schoolService = new SchoolServiceImpl();
     List<School> schoolList = schoolService.selectAllSchool();
     request.setAttribute("schoolList", schoolList);

     // 查询所有的公开课记录
     List<OpenCourse> openCourseList = openCourseService.selectOpenCourse(null);

     int pageSize = 3;// 每页记录个数
     int recordNum = openCourseList.size();// 总共查询的记录数

     int pageNum = recordNum / pageSize;// 一共可分几页
     if (recordNum % pageSize != 0) {
         pageNum++;
     }
     request.setAttribute("pageNum", pageNum);
     request.getRequestDispatcher("publicclass/selectClass.jsp").forward(request, response);

  }
  if (status.equals("second")) {
     response.setContentType("text/html;charset=utf-8");
     response.setCharacterEncoding("UTF-8");
     PrintWriter out = response.getWriter();

     Map<String, Object> map = new HashMap<String, Object>();
     String startDate = EncodingUtil.changeEncoding(request.getParameter("startDate"));
     if (!startDate.equals("")) {
         map.put("startDate", startDate);
     }
     String endDate = EncodingUtil.changeEncoding(request.getParameter("endDate"));
     if (!endDate.equals("")) {
         map.put("endDate", endDate);
     }
     String courseName = EncodingUtil.changeEncoding(request.getParameter("courseName"));
     if (!courseName.equals("")) {
         map.put("courseName", courseName);
     }
     String schoolId = EncodingUtil.changeEncoding(request.getParameter("school"));
     if (!schoolId.equals("")) {
         map.put("schoolId", schoolId);
     }
     String deptId = EncodingUtil.changeEncoding(request.getParameter("department"));
     if (!deptId.equals("")) {
         map.put("deptId", deptId);
     }
     String grade = EncodingUtil.changeEncoding(request.getParameter("grade"));
     if (!grade.equals("")) {
  ```

```
            map.put("grade", grade);
        }
        // 根据条件查询公开课记录
        List<OpenCourse> openCourseList = openCourseService.selectOpenCourse(map);

        // 根据网页传来的页码进行分页
        int pageSize = 3;// 每页记录个数
        int currPage =
Integer.parseInt(EncodingUtil.changeEncoding(request.getParameter("page")));// 获取当前页码
        int brginPage = (currPage - 1) * pageSize;// 分页开始索引
        int endPage = currPage * pageSize;// 分页结束索引
        if (endPage > openCourseList.size()) {// 限制索引使其不超过边际边际
            endPage = openCourseList.size();
        }
        List<OpenCourse> subOpenCourseList = openCourseList.subList(brginPage, endPage);
        // 使用阿里的插件，转为JSON字符串
        String jsonStr = JSON.toJSONString(subOpenCourseList);
        out.print(jsonStr);

    }
```

## （五）日历插件

- LayUI 日期插件
- 实例程序

```
<!-- 注意：需要官网下好组件包，解压放到自己的工程中 -->
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>使用 layDate 独立版</title>
        <script src="laydate/laydate.js"></script> <!-- 改成你的路径 -->
    </head>

    <body>
        <input type="text" id="test1" placeholder="请选择日期" >

        <script>
        //执行一个laydate实例
        laydate.render({
          elem: '#test1' //指定元素
        });
        </script>
    </body>
</html>
```

## （六）用户访问权限设置

- 使用过滤器

```
// 注：登陆时需要记录讲登陆成功的用户名记录到Session中
HttpSession session = request.getSession();// 获取Sesson
session.setAttribute("username", userName);
```

```
// 使用过滤器
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws IOException, ServletException {
    HttpServletRequest httpServletRequest = (HttpServletRequest)request;
    HttpServletResponse httpServletResponse = (HttpServletResponse)response;
    HttpSession sesson = httpServletRequest.getSession();

    if(sesson.getAttribute("username") == null) {
        httpServletResponse.sendRedirect(httpServletRequest.getContextPath() +
"/login.jsp");
    }else {
        chain.doFilter(request, response);
    }
}
```

- 前端相关程序

```
<!-- 实现自动登陆 -->
<script type="text/javascript">
    // 获取指定cookie的值
    function getCookie(cname){
        var name = cname + "=";
        var ca = document.cookie.split(';');
        for(var i=0; i<ca.length; i++) {
            var c = ca[i].trim();
            if (c.indexOf(name)==0) {
                return c.substring(name.length,c.length);
            }
        }
        return "";
    }

    // 获得cookie中存放的密码实现自动登陆
    function checkCookie()
    {
        var username=getCookie("username");
        var userpwd=getCookie("userpwd");
        $("#username").val(username);
        $("#userpwd").val(userpwd);
        // alert("username: "+ username);
        // alert("userpwd: "+ userpwd);

        if (username!="" && userpwd!="") {
            //location.href="AutoLoginServlet?username="+username+"&userpwd="+userpwd;
        } else {
          //alert("Please login !");
        }

    }
</script>
```

## （七）记住密码

- 使用Cookie

```
// 登陆验证
boolean result = userService.checkUser(userName, userPwd);
if (result) {
    String isRemember = EncodingUtil.changeEncoding(request.getParameter("isRemember"));
    // 记住密码
    if (isRemember.equals("true")) {
        // 创建两个cookie，分别用于存放用户名以及密码
        Cookie cookie1 = new Cookie("username", userName);
        Cookie cookie2 = new Cookie("userpwd", userPwd);

        // 指定Cookie绑定的路径，注意：这里指定的路径要求必须添加上项目名称request.getcontextpath()
        // cookie1.setPath(request.getContextPath() + "/username");
        // cookie2.setPath(request.getContextPath() + "/userpwd");

        // 设置Cookie有效期，参数为一个整型值，单位为秒
        // 该值大于0，表示将Cookie存放到客户端的硬盘
        // 该值小于0，与不设置效果相同，会将Cookie存放到浏览器缓存中
        // 该值等于0，表示Cookie一生成，马上失效
        cookie1.setMaxAge(60 * 60 * 24 * 10);// 设置Cookie有效期为10天
        cookie2.setMaxAge(30 * 1);// 设置Cookie有效期为1小时
        // 向响应中添加Cookie
        response.addCookie(cookie1);
        response.addCookie(cookie2);
    }

    HttpSession session = request.getSession();// 获取Sesson
    session.setAttribute("username", userName);
    response.sendRedirect("main.jsp");// 重定向页面
} else {
    response.sendRedirect("login.jsp");// 重定向页面
}
```

## （八）网页实时显示时间

- 相关程序

```
<script type="text/javascript">
/**
*实时显示系统时间
*/
function getLangDate(){
        var dateObj = new Date(); //表示当前系统时间的Date对象
        var year = dateObj.getFullYear(); //当前系统时间的完整年份值
        var month = dateObj.getMonth()+1; //当前系统时间的月份值
        var date = dateObj.getDate(); //当前系统时间的月份中的日
        var day = dateObj.getDay(); //当前系统时间中的星期值
        var weeks = ["星期日","星期一","星期二","星期三","星期四","星期五","星期六"];
        var week = weeks[day]; //根据星期值，从数组中获取对应的星期字符串
        var hour = dateObj.getHours(); //当前系统时间的小时值
        var minute = dateObj.getMinutes(); //当前系统时间的分钟值
        var second = dateObj.getSeconds(); //当前系统时间的秒钟值
        //如果月、日、小时、分、秒的值小于10，在前面补0
        if(month<10){
        month = "0"+month;
        }
```

```
        if(date<10){
        date = "0"+date;
        }
        if(hour<10){
        hour = "0"+hour;
        }
        if(minute<10){
        minute = "0"+minute;
        }
        if(second<10){
        second = "0"+second;
        }
        var newDate = year+"年"+month+"月"+date+"日 "+week+" "+hour+":"+minute+":"+second;
        document.getElementById("systemdate").innerHTML = "系统公告: [ "+newDate+" ]";
        setTimeout("getLangDate()",1000);//每隔1秒重新调用一次该函数
}
</script>

// 进入页面就调用
<body onload="getLangDate()">
```

# 三、FAQ

## （一）经过servlet跳转后jsp页面的样式消失

> 当在jsp中引入css时，如果其相对路径相对于当前jsp文件的，而在一个和这个jsp的路径不一样的servlet中forward这个jsp时，就会发现这个css样式根本没有起作用。 这是因为在servlet中转发时css的路径就是相对于这个servlet的相对路径而非jsp的路径了。 参考链接 参考链接

- 解决方法 (1)引入下列代码

```
<%
    // getContextPath() 返回当前页面所在的工程的物理路径（项目名称+/）
    String path = request.getContextPath();

    // basePath是拼装得到的当前网页的相对路径的。
    // request.getScheme() 返回当前页面使用的协议，http 或是 https
    // request.getServerName() 返回当前页面所在的服务器的名字，在就是localhost
    // request.getServerPort() 返回当前页面所在的服务器使用的端口
    String basePath = request.getScheme()+"://"
            +request.getServerName()+":"
            +request.getServerPort()+path+"/";
%>

<!-- 为页面上的所有链接规定默认地址或默认目标 -->
<base href="<%=basePath%>">
```

(2)将路径改为根目录下的绝对路径（因为有`<base>`标签所以自动加上定义好的相对路径）

```
<!-- 更改前 -->
<link href="../css/style.css" rel="stylesheet" type="text/css" />
<!-- 更改后 -->
<link href="css/style.css" rel="stylesheet" type="text/css" />
```

(3)如果没有写标签就需要在所有的css样式、js或者图片引用的路径前加上`<%=basePath%>`

```
<!-- 更改前 -->
<link href="../css/style.css" rel="stylesheet" type="text/css" />
<!-- 更改后 -->
<link href="<%=basePath%>/css/style.css" rel="stylesheet" type="text/css" />
```